Microcomputadores

Arquitetura • Projeto • Programação

PAULO BIANCHI
MILTON BEZERRA

LTC LIVROS TECNICOS E CIENTIFICOS EDITORA S.A.

Microcomputadores

Arquitetura • Projeto • Programação

PAULO BIANCHI FRANÇA

- *Analista do Núcleo de Computação Eletrônica — UFRJ*
- *Professor Adjunto do Instituto de Matemática — UFRJ*
- *PhD em Engenharia Elétrica e Ciências de Computação — Universidade da Califórnia — Berkeley*

MILTON DE ALBUQUERQUE BEZERRA

- *MSc em Engenharia de Sistemas de Computação — COPPE — UFRJ*
- *Analista do Núcleo de Computação Eletrônica — UFRJ*
- *Professor Assistente do Instituto de Matemática — UFRJ*

RIO DE JANEIRO
SÃO PAULO

Membros da Comissão de Computação da LTC: professor Ysmar Vianna; professor Donaldo de Souza Dias e professor Luís de Castro Martins.
Coordenador da obra: professor Ysmar Vianna — Núcleo de Computação da UFRJ
Revisor do texto: Waldyr dos Santos Dias
Revisor de provas: Carlos André Sucupira
Diagramação e Paginação: José Mesquita
Capa: AG Comunicação Visual Assessoria e Projetos Ltda.

CIP-Brasil. Catalogação-na-fonte
Sindicato Nacional dos Editores de Livros, RJ.

França, Paulo Bianchi.
F882m Microcomputadores: arquitetura, projeto, programação / Paulo Bianchi França, Milton de Albuquerque Bezerra. – Rio de Janeiro: LTC – Livros Técnicos e Científicos Editora S.A., 1983.

Apêndices
ISBN

1. Microcomputadores I. Bezerra, Milton de Albuquerque II. Título

CDD – 001.64
83-0541 CDU – 681.3

ISBN: 85-216-0321-5

Direitos reservados por:
LTC — LIVROS TÉCNICOS E CIENTÍFICOS EDITORA S.A.
Av. Venezuela, 163 — 20220
Rio de Janeiro, RJ
1983
Impresso no Brasil

À
Áurea Teresa, Andréa e Ricardo
P.B.F.

À
Lucia Regina e Cristiana
M.A.B.

APRESENTAÇÃO

O texto se destina aos leitores que já sabem o que é um computador e o que é um programa. Não é necessário nenhum conhecimento de eletrônica. Destina-se a aprofundar os conhecimentos relativos à construção de computadores e programação a nível de linguagem assembler e de máquina.

Acreditamos que tanto os profissionais de computação quanto os entusiastas do computador caseiro, possam aqui encontrar um meio de satisfazer à sua *ânsia de aprofundamento* pois procuramos uma exposição bastante acessível.

O texto se originou de notas de aulas de vários cursos referentes a projeto com microprocessadores ministrados pelo Prof. Paulo Bianchi na Universidade de Santa Clara, Califórnia e na Universidade Federal do Rio de Janeiro, bem como dos cursos referentes a programação de computadores ministrados pelo Prof. Milton Bezerra na Universidade Federal do Rio de Janeiro. Utilizamos os microprocessadores da família 8080 (8080, 8085 e Z80) por serem os mais populares; no entanto, procuramos sempre reforçar os conceitos básicos de tal forma que o leitor possa, sem grande dificuldade, acompanhar o manual de outros sistemas.

O Cap. 1 é introdutório visando nivelar os conhecimentos dos leitores Os Caps. 2 e 5 são relativos a "hardware" e podem ser usados independentemente como texto de um curso de introdução à arquitetura de computadores, como de fato tem ocorrido. O Cap. 2 lida apenas com conceitos, explicando como funcionam os circuitos e como são construídos os computadores a partir de circuitos mais simples. Este capítulo não depende de nenhum microprocessador real. O Cap. 5 explora a realidade dos microprocessadores usando, principalmente, o micro 8085.

Os Caps. 3 e 4 referem-se à programação e podem também ser usados independentemente como texto de cursos envolvendo programação assembler e linguagem de máquina. O Cap. 3 explica o funcionamento dos processadores 8080, 8085 e Z80 e descreve as instruções dos mesmos. Este capítulo está organizado de modo a servir também como manual de referência para o leitor

mais experiente. O Cap. 4 desenvolve o assunto de programação com mais profundidade explicando a linguagem assembler e apresentando diversos exemplos.

Todos os capítulos são seguidos de uma pequena série de exercícios que encorajamos fortemente os leitores a resolver.

Finalmente, devemos notar que o assunto de que tratamos é muito popular e há uma grande necessidade de ser divulgado em grande escala. Assim sendo, visando facilitar o trabalho dos instrutores, os autores também prepararam material didático que se encontra à disposição dos interessados.

OS AUTORES

PREFÁCIO

Foi grande a minha satisfação quando recebi a grata incumbência de apresentar este texto "MICROCOMPUTADORES", Arquitetura, Projeto e Programação, não somente pelo seu conteúdo técnico, muito apropriado para os nossos dias, mas, também, por serem seus autores jovens ainda, que se iniciaram em informática no Núcleo de Computação Eletrônica — UFRJ e foram meus colaboradores em outras épocas.

Paulo Bianchi França, ainda como estudante ingressou, em 1967, no DCC-COPPE, tendo me auxiliado na revisão do FORTRAN Monitor, naquele ano. Milton de Albuquerque Bezerra iniciou-se no NCE — UFRJ, em meados da década de 70 e, hoje, participa ativamente junto à direção daquele Núcleo.

Voltando ao texto propriamente dito, o assunto em pauta é tratado de maneira objetiva e apropriada, visando o leitor que tenha, pelo menos, algum conhecimento de programação (FORTRAN, PASCAL, ALGOL, COBOL etc.).

Os mercados a atingir, que antevejo, de grande utilidade, serão os seguites:

1) Cursos de Informática, onde poderia ser usado como livro-texto nas cadeiras de Arquitetura de Computadores e Programação Assembler.

2) Os profissionais de Informática do tipo Analista e Programador, que os aproximarão mais do *hardware*, lembrando que cada vez mais desaparece o "gap" entre o *hardware* e o *software*.

3) O pessoal amador de Informática que já monta "micro" caseiro, pois iria satisfazer a curiosidade do *hardware*.

4) Aos Engenheiros de Controle, uma vez que o uso dos microcomputadores está penetrando, decisivamente, nessa área da Engenharia.

Alegro-me, também, ao saber que a LTC se propôs a editar este texto, assim como a OEA irá patrocinar uma versão em castelhano.

Tércio Pacitti

SUMÁRIO

OS MICROCOMPUTADORES E SEU MUNDO

1.1. COMPUTADORES ONTEM E HOJE

O Novo Universo

Durante a década de 60 os computadores começaram a se integrar na sociedade brasileira, as grandes empresas sediadas no país iniciando suas atividades de automação com grandes expectativas. Desde então, os computadores foram se tornando cada vez mais populares e no início dos anos 70 ainda se acreditava que os computadores continuariam a ser equipamentos complexos, grandes e caros.

Mas isso mudou. Com a progressiva evolução da tecnologia, que tem resultado em graus de miniaturização e integração dificilmente imagináveis no passado, os computadores têm se popularizado cada vez mais a ponto de se situarem dentro do poder aquisitivo individual.

O novo espectro de aplicações coberto pelo computador modifica, de modo relevante, o universo que nos cerca trazendo uma nova ordem de coisas. Dois aspectos dessa mudança são objetivamente tratados no texto:

— Integração *software-hardware*;
— Nova metodologia de trabalho.

Um terceiro aspecto são os computadores caseiros para entretenimento e educação. Apesar de não ter sido este um dos objetivos iniciais, o leitor interessado neste assunto disporá de farto material nos capítulos seguintes.

Integração *"Software-Hardware"*

A distância entre o equipamento (*hardware*) e sua programação (*software*) tem diminuído progressivamente. O usuário (ainda que profissional) de computador sempre encarou o equipamento como fixo e imutável; cabendo-lhe, apenas, desenvolver ou projetar sua programação. Quando muito, o usuário poderia escolher um equipamento que fosse mais adequado à sua necessidade.

Atualmente, o entendimento da problemática de projeto de equipamentos se tornou bastante acessível, permitindo que o próprio usuário idealize um equipamento adequado seja através de construção própria ou por encomenda a um fabricante.

Nova Metodologia de Trabalho

O uso de computador tem sido orientado para o modelo centralizado, onde um computador atende a todos os serviços. O usuário pode ter que se deslocar fisicamente até as vizinhanças do computador ou usá-lo remotamente através de um terminal.

Esta centralização decorreu de que, no passado, se observou que centralizar era mais econômico do que distribuir.

Atualmente, com computadores baratos, a distribuição tem se apresentado como uma alternativa atraente. Além do baixo custo da aquisição dos microcomputadores, nota-se que o usuário sente-se mais à vontade tendo seu próprio computador, dependendo menos da instalação central e concentrando mais responsabilidade sob seu próprio controle.

Não só o analista de sistemas deve se conscientizar dessa mudança, mas também os profissionais usuários de computador. Mais ainda, o universo está em constante mutação; outras mudanças ainda estão por ocorrer. Para antecipar-se é fundamental conhecer o estado e o rumo da tecnologia atual.

1.2. MICROPROCESSADORES E MICROCOMPUTADORES

Microprocessador

No início da década de 70 a tecnologia eletrônica produziu um novo elemento no mundo dos computadores: o microprocessador. O microprocessador contém o circuito de aritmética, controle e execução de um computador (isto é, a unidade de processamento) dentro de uma embalagem minúscula (Fig. 1.1).

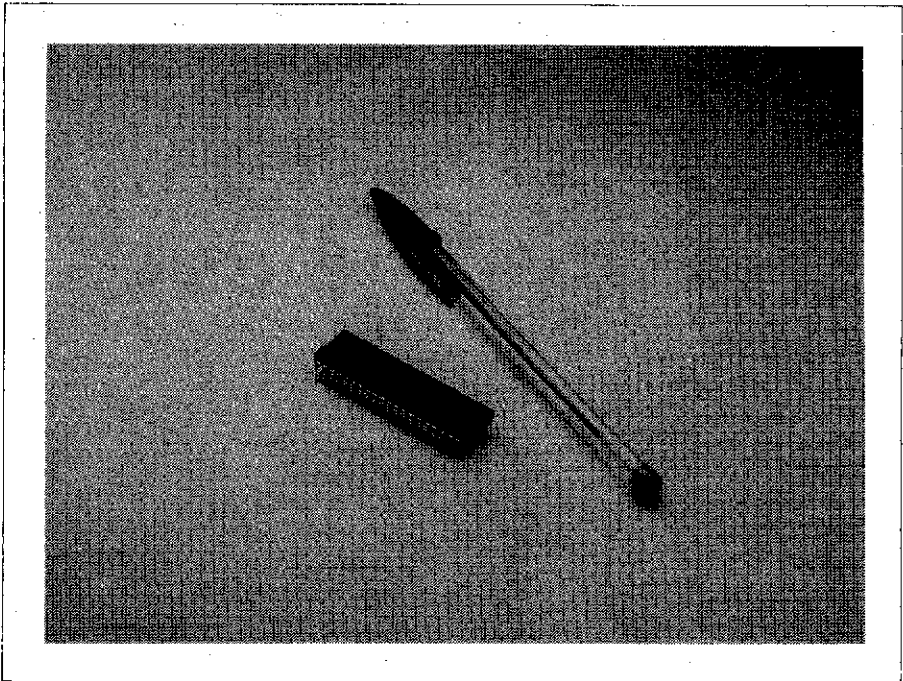



Fig. 1.1 — Microprocessador.

Apesar de seu tamanho, que justifica o prefixo "MICRO", o microprocessador já era mais rápido e possante que os mais populares computadores de segunda geração (IBM-1401, Burroughs B-200 etc.). No entanto, não foram aproveitados de imediato como computadores de pequenas dimensões; inicialmente, foram empregados em substituição a circuitos eletrônicos em máquinas de calcular e em equipamentos periféricos de computadores (como impressoras, leitoras, terminais de vídeo etc.).

As características marcantes dos microprocessadores são:

— Baixo custo
— Pequenas dimensões e peso
— Baixo consumo de energia
— Generalidade de uso.

Microcomputador

Somente na metade da década de 70 começaram a surgir os primeiros microcomputadores, isto é, computadores pequenos que usavam os microprocessadores como unidade de processamento (ver Fig. 1.2). Logo se descobriu que o mercado para microcomputadores era bem maior do que o inicialmente previsto; além daqueles que adquiriam o sistema para se educar ou pesquisar, começaram a aparecer os profissionais (que descobriam que podiam comprar um computador!), e os pequenos negociantes.

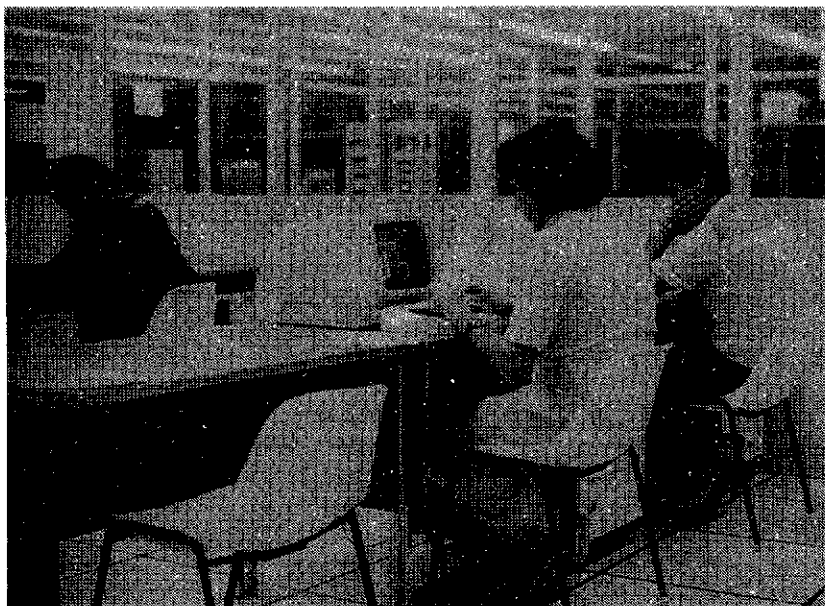



Fig. 1.2 — Microcomputador

Mas não acaba aí a escalada dos microcomputadores. Também as grandes empresas começaram a se interessar pelos micros notando que em diversas ocasiões um microcomputador é mais vantajoso do que um terminal para um sistema central. Esta última descoberta, que ainda se encontra em desenvolvimento, reformula diversos conceitos em processamento de dados exigindo mudanças fundamentais no estilo de projeto de sistemas.

Computadores e Microcomputadores. O microcomputador é, como o nome diz, um computador pequeno; existem alguns conceitos que diferem dos demais computadores e que examinaremos a seguir. Classificamos as diferenças como: limitações do microprocessador, limitações do sistema microcomputador e seu fabricante, e limitações do *software*.

Limitações do Microprocessador

A maioria dos fabricantes de microcomputadores no Brasil se utiliza de unidades de processamento muitíssimo semelhante; isto ocorre porque:

1. os microprocessadores (que constituem a CPU) não são feitos sob medida para cada fabricante (como ocorre com computadores maiores) mas são vendidos para qualquer pessoa ou fabricante interessado.

2. Dentre os diversos modelos de microprocessadores existe um conjunto, denominado a família 8080 (será estudada no texto), que alcançou grande índice de popularidade e os fabricantes nacionais, em sua grande maioria, se utilizam de micros desta família.

O fato de se ter um mesmo processador implica em ter o mesmo conjunto de instruções, a mesma velocidade etc. Fatores estes que não diferirão de um microcomputador para outro (exceto, é claro, usando microprocessadores diferentes).

Comparativamente aos computadores, os microprocessadores da família 8080 impõem as seguintes restrições:

1. Operações caracter a caracter. Os micros operam, em geral, com 1 caracter de cada vez; computadores de maior porte podem dispor de instruções capazes de operar com um conjunto de caracteres; tais como mover 100 caracteres de um lugar para outro na memória. O micro fará ao invés disso, 100 movimentações de 1 caracter (o que será mais demorado).

2. Aritmética. Em geral os computadores dispõem de operações aritméticas de soma, subtração, multiplicação e divisão de números inteiros (ponto fixo) podendo também dispor de operações para números reais (ponto flutuante). Cada operação, dependendo do computador, pode considerar números de 16 bits (caso típico de minicomputadores), 32 ou 64 bits (máquinas de grande porte) (ver Seç. 1.5 — Representação de Dados).

O microprocessador opera com números de 8 bits, podendo, é claro, somar números maiores desde que em etapas de 8 bits.

Mais ainda: os micros só operam com números inteiros e *não* têm instruções de multiplicar nem dividir. (Neste ponto, tudo parece estar perdido!) Não esqueçamos, no entanto, que usando essas operações elementares é possível implementar as demais; assim tais operações são resolvidas por programação tendo como única conseqüência sensível uma demora no tempo de execução.

3. Espaço de endereçamento. Os microcomputadores da família 8080 têm condição de endereçar 64K ($1k = 1024$) bytes de memória. Computadores maio-

res têm atualmente condição de endereçar até 16 milhões de bytes. Convém lembrar que a maioria dos minicomputadores também se restringem a 64K e que, na década de 60, computadores grandes "pintavam e bordavam" com muito menos.

4. Tipos de endereçamento. Para operar dados contidos na memória, o processador precisa fornecer o endereço a fim de obter o dado. Existem várias formas de fornecer o endereço visando facilitar o trabalho do programador (uma discussão mais completa de endereçamento será feita no capítulo sobre programação).

A variedade de modos de endereçamento é reduzida nos microcomputadores inexistindo o modo de endereçamento indireto. O endereçamento por índice, muito útil para operar sobre vetores ou cadeias de dados, só está presente nos micros do tipo Z80.

5. Velocidades. As instruções de um micro podem demorar mais ou menos de acordo com o tipo da instrução e eventualmente com o modelo de microprocessador utilizado (alguns são mais rápidos) porém, podemos admitir um valor médio ordem de 3 microssegundos (0,000003 segundos) por instrução. Este tempo não é muito diferente em computadores maiores; o outrora popularíssimo IBM 1130 tinha um tempo médio da ordem de 7 microssegundos. Vamos lembrar porém que as instruções do micro são mais simples sendo necessário executar mais instruções para executar a mesma tarefa.

Outra velocidade que devemos considerar é a da memória. Neste ponto os micros só têm a perder para máquinas de grande porte que usam artifícios para aumentar a velocidade efetiva da memória. Via de regra os micros operam com memórias cujo tempo de acesso varia entre 200 e 450 nanossegundos (0.000000450 segundos).

6. Suporte de multiprogramação. Os micros não oferecem suporte para multiprogramação, isto é, execução de mais de um programa ao mesmo tempo. Não há, em princípio, como evitar que um usuário interfira com a execução do outro, destruindo dados, programas etc. Isto porém, não tem impedido o aparecimento de sistemas multiprogramados onde este suporte é provido "artificialmente".

7. Suporte para memória virtual. Os microprocessadores (da família 8080) não dispõem de nenhum suporte para memória virtual.

8. Suporte de comunicação. Todos os microprocessadores têm condições de suportar comunicação de dados.

Limitações do Sistema Microcomputador

As limitações do sistema são aquelas impostas pelo fabricante do microcomputador seja pelo projeto do sistema como um todo, seja pela fabricação, seja pela comercialização. Tais limitações se fazem sentir, basicamente, pela limitação dos equipamentos periféricos, observando-se, em geral, o seguinte:

— a quantidade de periféricos por sistema é pequena
— existência de periféricos típicos.

Um computador tem condições de gerenciar um número grande de periféricos como, por exemplo, 8 unidades de fita, 16 de disco, 5 impressoras, 80 terminais etc. O microcomputador, em geral, não tem condição de gerenciar tantos periféricos simultaneamente.

Existem, por outro lado, periféricos típicos, que são os mais utilizados com microcomputadores tais como: o teclado, o vídeo, disco e impressora; em alguns casos encontram-se gravadores cassette (em substituição ao disco devido a seu menor custo) e ligação para uma TV comum (em substituição ao vídeo).

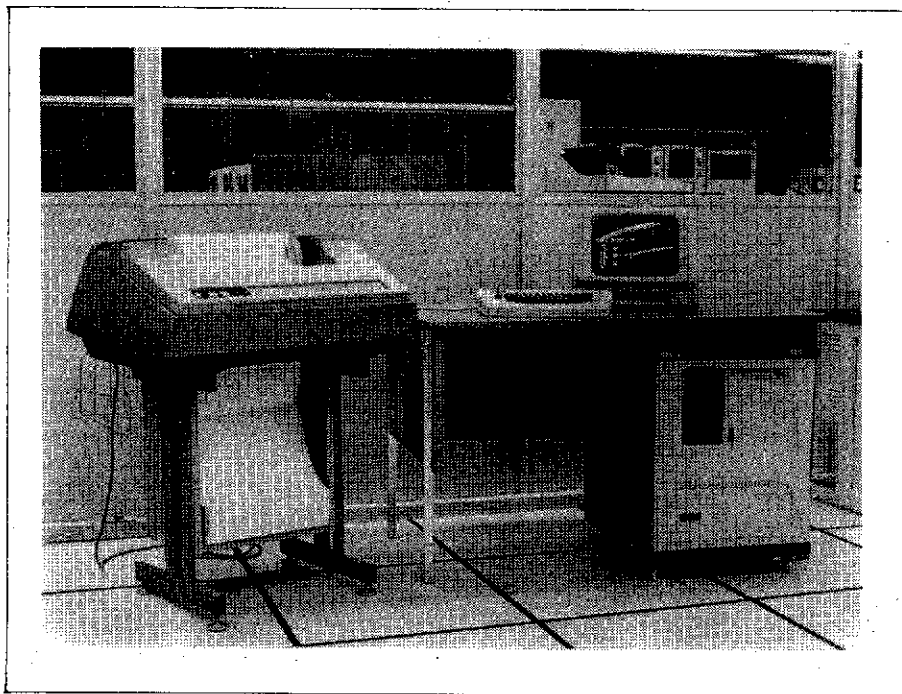



Fig. 1.3 — Periféricos típicos.

Os discos mais usados em microprocessadores são os discos flexíveis (floppy disks) também chamados diskettes. Mais recentemente foram introduzidos os discos Winchester.

Disco flexível. Os discos flexíveis, que são os mais populares e econômicos, funcionam dentro de um envelope de papel havendo dois métodos para gravação: densidade simples que permite gravar até 256K bytes em uma face do disco e densidade dupla que permite gravar até 512K bytes em uma face. A densidade simples ou dupla refere-se ao modo de gravação, sendo uma característica do equipamento e não do diskette. Existem diskettes que só podem ser gravados em uma das faces e outros modelos em que ambas as faces podem ser utilizadas.

A unidade de disco (isto é, o equipamento que lê ou grava) pode ser de uma ou duas faces. Nas unidades de duas faces, o micro pode ter acesso a dados gravados em ambas as faces do diskette; na unidade simples pode-se usar um diskette de duas faces como num toca-discos, sendo necessário virar o diskette para se ter acesso a outra face.

Em geral, um sistema microcomputador pode trabalhar com até 4 unidades de diskette resultando em até 4 milhões de bytes (dupla densidade, dupla face-

DDDF) de armazenamento em linha. Este total, no entanto, depende do fabricante e do modelo do microcomputador.

Disco Winchester. Os discos flexíveis apesar de convenientes, têm constituído um sério gargalo para as aplicações onde se requer armazenamento de grande número de dados sendo comum o uso de minicomputadores ao invés de micros somente devido à disponibilidade de discos de maior capacidade.

Um outro tipo de disco que surgiu recentemente, é o disco do tipo winchester que consta de um conjunto de discos superpostos e envoltos por um plástico lacrado.

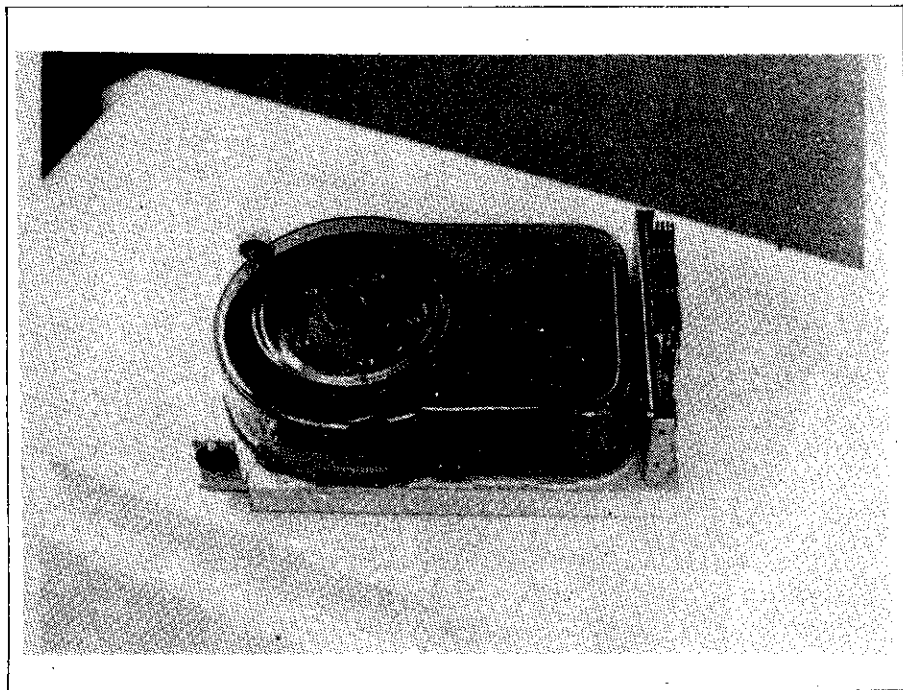



Fig. 1.4 — Disco winchester.

Existem modelos capazes de armazenar 5 Megabytes ou 10 Megabytes. A grande desvantagem dos discos Winchester é que não são removíveis (não podem ser trocados pelo operador como os diskettes).

Limitações *Software*

Um terceiro tipo de limitação que acompanha qualquer computador é o conjunto de programas disponíveis, que oferecem aos usuários maiores ou menores facilidades para resolver seus problemas. Dentre tais programas, devemos destacar: Sistemas Operacionais, Linguagens de alto nível e programas aplicativos e utilitários.

Sistema Operacional. O sistema operacional é um conjunto de programas que acompanha o computador tendo por objetivo gerenciar os recursos da

máquina (espaço em disco, memória etc.) e prover uma comunicação mais simples entre o usuário e o computador. Tipicamente o usuário conversa com o sistema operacional através do teclado fornecendo comandos tais como: apagar um conjunto de dados, executar um programa, armazenar dados em disco etc.

O sistema operacional executa esses comandos e informa ao usuário, através do vídeo, qualquer anormalidade encontrada (acabou espaço em disco etc.).

No mundo dos microcomputadores de uso profissional observa-se outra particularidade importante no Brasil: existem duas famílias de sistemas operacionais: SOM (Cobra) e CP/M (demais fabricantes).

CP/M. O sistema CP/M (Control Program/Microcomputer) foi desenvolvido pela Digital Research da Califórnia, EUA. É um sistema simples com duas características importantes: abrangência e portabilidade.

Quando começou o movimento de microcomputadores, vários fabricantes pequenos começaram a lançar seus sistemas acompanhados de sistemas operacionais muito limitados que não ofereciam as facilidades desejáveis. Isto porque, sendo empresas pequenas, vendendo um número relativamente baixo de sistemas não apostaram na lucratividade do sistema operacional. Nessa época a idéia explorada pela Digital Research, também uma pequena empresa, foi desenvolver um sistema operacional mais completo e que pudesse ser usado não somente para um modelo de microcomputador mas sim por diversos, capturando o mercado que foi aberto pelas diversas empresas fabricantes. Juntando essa idéia com um preço relativamente baixo de comercialização (US$ 150,00) o CP/M resultou em ser o único sistema que atendia às necessidades dos usuários de uma grande variedade de equipamentos, além disso custando pouco.

A partir daí o CP/M tornou-se um padrão de fato para microcomputadores tendo uma penetração limitada somente no mercado de equipamentos tipo Apple e TRS-80 fabricados em muito grande quantidade e fornecidos com programação do fabricante.

O padrão gerado pelo CP/M teve conseqüências importantes: já que o programa desenvolvido usando CP/M pode ser usado em qualquer outro equipamento usando CP/M, o mercado de programas foi muito ampliado, resultando no aparecimento de uma grande variedade de programas a baixo custo.

Esta motivação levou vários fabricantes nacionais a optar pelo uso do sistema CP/M ou sistemas derivados do CP/M ao invés de desenvolverem cada um seu próprio sistema o que seria muito mais dispendioso e não traria vantagem alguma aos usuários.

Posteriormente, a própria Digital Research iniciou a comercialização de outros sistemas da mesma família CP/M: o MP/M que suporta multiprogramação com vários usuários trabalhando concorrentemente e o CP/NET que permite comunicação entre vários sistemas operando com CP/M.

Estes sistemas constituem uma família porque, para o usuário, o sistema continua com a mesma estrutura, executando os mesmos programas oferecendo apenas algumas facilidades a mais.

SOM. A Cobra (Computadores Brasileiros S.A.) foi o único fabricante nacional de microcomputadores que optou pelo desenvolvimento de um sistema operacional próprio. Tendo em vista que a Cobra é a empresa de maior porte da área, as despesas decorrentes são mais facilmente absorvidas e justificáveis.

Em geral um sistema próprio aumenta o vínculo do usuário com o fabricante; porém, outro argumento importante nessa decisão foi, certamente, o objetivo de desenvolver um sistema nacional.

UNIX[1], *XENIX, IDRIS.* Outro sistema que possivelmente ocupará posição de destaque no futuro é o UNIX. O UNIX foi desenvolvido por dois pesquisadores dos laboratórios Bell para computadores da série Digital PDP-11. Sendo um sistema de fácil aprendizado e contando com muitas facilidades para desenvolvimento, começou a despertar o interesse de usuários de PDP-11.

Posteriormente, a Bell começou a distribuir o sistema a preço de custo para as universidades americanas e a comercializá-lo com as demais empresas. Essa política vem resultando em que o UNIX está se tornando um padrão "de fato" na universidade e o aparecimento de uma "fartura" de programas em torno do mesmo (de modo semelhante ao que ocorre com o CP/M para microcomputadores).

Atualmente o âmbito do UNIX começa a transcender a família Digital (PDP-11, VAX) havendo implementações para microcomputadores de 8 e de 16 bits. Tudo levando a crer que não só será o UNIX o padrão para microcomputadores no futuro próximo, como também, será um dos primeiros sistemas operacionais transcendendo a barreira dos fabricantes.

Apesar de todo esse sucesso, o UNIX é quase desconhecido no Brasil porque, baseada em temores legais (possíveis interpretações da lei do segredo industrial), a Bell tem se recusado a ceder ou vender o UNIX para instituições brasileiras. Existem fontes alternativas que também comercializam sistemas equivalentes ao UNIX porém os problemas comerciais ainda não foram superados.

Linguagens de alto nível: Outro suporte de programação de importância fundamental são os programas compiladores para linguagens de alto nível. Tais programas "traduzem" programas feitos pelo usuário em uma linguagem mais inteligível (linguagem de alto nível) para a linguagem conhecida pelo computador (linguagem de máquina).

De um modo geral, os microcomputadores de uso profissional dispõem de uma variedade grande de compiladores: BASIC, FORTRAN, COBOL, PASCAL, PL/I, por exemplo. A fartura de programação deve-se, em grande parte, à padronização que levou um grande número de empresas a se interessar pelo assunto.

Pode-se dizer que a linguagem mais popular para os microcomputadores é o BASIC havendo versões diferentes em uso.

Em algumas versões (TRS-80, DISMAC-D8000, PROLOGICA CP500 etc.) o programa do usuário é guardado na memória exatamente como foi feito (não é compilado, traduzido para linguagem de máquina). O programa é analisado comando a comando e executado por um programa interpretador. A execução é muito mais lenta porém a programação é muito mais fácil pois tudo se passa como se o computador "falasse" em BASIC. Neste caso não temos um "compilador" mas sim um "interpretador" BASIC.

Em outras versões o programa do usuário é compilado (traduzido) para então ser executado. Deste modo a execução é mais rápida porém a depuração é mais demorada. Tendo em vista que as diferenças de programação são mínimas chega a ser aconselhável que o usuário disponha das duas versões usando

---

[1] UNIX é marca registrada da Bell Laboratories, Inc.

o interpretador enquanto desenvolve o programa e o compilador para a versão final.

Para os sistemas CP/M o interpretador MBASIC (Microsoft-Basic) e o compilador CBASIC (Commercial Basic) são os mais comuns. Ambos suportam variáveis com nomes de até 31 caracteres, números reais com 14 dígitos de precisão, amplas facilidades para manuseio de cadeias alfanuméricas, formatação de entrada/saída, acesso seqüencial e direto a arquivos.

Aplicativos e utilitários. Programas aplicativos e utilitários são aqueles diretamente usados pelo usuário para resolver seus problemas do dia-a-dia. A fartura de programas dessa categoria é ainda maior; porém, tendo em vista que a possível variedade é maior do que as demais categorias de programas, conclui-se que o mercado está muito longe de ser saturado.

1.3. MICROELETRÔNICA

O elemento básico para a construção de circuitos de computador é o transistor. Inicialmente os transistores eram fabricados e vendidos individualmente; para se montar um circuito era necessário adquirir os transistores e interligá-los da maneira mais adequada.

O Circuito Integrado

Posteriormente, passou a ser possível fabricar na mesma embalagem diversos transistores já interligados de modo adequado; daí o nome de circuito integrado já que transistores e interligação estão integrados em um único circuito, fabricados e embalados de uma só vez.

Classificação Quanto à Densidade

Os primeiros circuitos integrados continham poucos transistores por embalagem. Posteriormente, a tecnologia foi evoluindo e permitindo que cada vez mais transistores fossem colocados dentro de uma embalagem. O número de transistores por embalagem é o que se chama de densidade do circuito e usa-se a seguinte nomenclatura aproximada para classificar os circuitos quanto à sua densidade:

SSI — até 10
MSI — até 500
LSI — até 50000
VLSI — até 500000

As siglas SSI, MSI, LSI e VLSI significam, respectivamente, integração em pequena escala (Small Scale Integration), integração em média escala (Medium Scale Integration), integração em alta escala (Large Scale Integration) e integração em muito alta escala (Very Large Scale Integration).

Observe-se também que esta classificação não é rígida e deve ser encarada mais como gíria do que como formalismo.

Classificação Quanto à Tecnologia

Os circuitos integrados podem ser fabricados usando-se vários tipos de tecnologia, cada tipo podendo obter essencialmente os mesmos resultados porém cada um apresentando características próprias. Assim, as características normalmente envolvidas: densidade, velocidade, consumo de energia do circuito resultante, dependerão da tecnologia empregada.

As tecnologias que nos interessam são:

— as famílias MOS
— CMOS
— BIPOLAR

Existem várias tecnologias empregando *M*etal, *O*xido e *S*emicondutor (daí a sigla MOS) que resolvemos denominar coletivamente por família MOS. A família MOS tem como características gerais permitir uma densidade extremamente alta, velocidade de operação razoável e consumo de energia também razoável. Esta tecnologia é a mais comumente usada nos microprocessadores atuais.

A tecnologia CMOS, que difere ligeiramente das tecnologias da família MOS tem, atualmente, recebido muita atenção devido à característica de resultar em um consumo de energia extremamente baixo. Inicialmente, o circuito CMOS tinha velocidade mais baixa e, aproximadamente, metade da densidade dos circuitos MOS porém, atualmente, as deficiências são bem menos sérias: apesar de que a velocidade e densidade CMOS não possam se igualar às MOS, estão se aproximando bastante desta. A tecnologia CMOS é comumente usada em calculadores e instrumentos portáteis devido ao seu baixo consumo energético.

Finalmente, a tecnologia bipolar, bastante diferente das demais, tem como grande vantagem a alta velocidade de operação e como duas grandes desvantagens uma baixíssima densidade e consumo de energia excessivamente alto.

1.4. MICROCOMPUTADORES E O PROFISSIONAL

Devido ao seu baixo custo e facilidade de uso, o microcomputador trará implicações muito grandes na vida profissional de diversas especialidades. Destacamos aqui as implicações para os profissionais de tecnologia (engenharia, matemática, arquitetura etc.), profissionais de computação, profissionais de gerência e profissionais de ensino.

O Profissional de Tecnologia

Este profissional está na linha direta de impacto do microcomputador. Da mesma forma que os calculadores eletrônicos tornaram obsoletas as réguas

de cálculo, os microcomputadores absorverão grande parte do trabalho atualmente feito com auxílio dos calculadores resultando numa economia de tempo do profissional. Mesmo o calculador programável exige que o profissional pense e resolva quais as operações e procedimentos a serem executados.

O microcomputador pode executar tarefas muito mais complexas exigindo muito menos atenção do profissional.

Para completar, a variedade de equipamentos que pode ser ligada a um microcomputador é muito grande podendo permitir mais eficiência na comunicação com o profissional (por exemplo, através de gráficos) e, mais importante, permitindo a troca de informações entre computadores através de discos, fitas ou mesmo linhas telefônicas.

O Profissional de Computação

Os profissionais de computação (analistas e programadores) também terão sua atividade alterada pelo microprocessador. Atualmente, tais profissionais, juntamente com "o computador" da empresa fazem parte de um "Centro de Processamento de Dados" e são, normalmente, os únicos entendidos em computação dentro da empresa.

O baixo custo e a facilidade de uso dos microcomputadores vai acarretar, sem sombra de dúvida, uma proliferação desses equipamentos em toda a empresa. Haverá então um vasto contingente com conhecimentos razoáveis, embora superficiais, de computação que virão participar intensamente das atividades de processamento de dados na empresa.

Dessa forma, os profissionais de computação receberão novas atribuições, que poderão até resultar em novas especialidades visando principalmente assistir à nova comunidade no uso de seus equipamentos.

O Profissional de Gerência

O uso do computador para assistir a gerência de mais alto nível, já está consagrado. Sistemas de informações gerenciais estão implantados na maioria das empresas, gerando relatórios para dar subsídios às decisões de alto nível.

O microcomputador permite uma mudança na escala de operação do sistema, permitindo aos diversos níveis de gerência dispor de seu próprio sistema de suporte a decisões. Dessa forma, além de colocar essa ferramenta ao alcance de mais pessoas, pode-se também aumentar a agilidade dos sistemas de informação já que informações poderão ser coletadas e recuperadas com apoio dos microcomputadores contornando o processo manual.

O Profissional de Ensino

Finalmente, outro tipo de profissional que será severamente afetado pelos microcomputadores é o profissional de ensino. O assunto Computadores na educação tem recebido crescente atenção, fruto, é claro, da redução no custo de computação causada pelos microprocessadores.

Devemos destacar dois aspectos interessantes do computador na educação:

— O computador como *objeto* do ensino;
— O computador como *ferramenta* de ensino.

Por computador como objeto do ensino entendemos o ensino de programação e uso de computadores. Devido aos custos decrescentes já se encontra o microcomputador na sala de aula do 1º grau onde os alunos aprendem com sucesso a programá-lo.

Mais interessante e de maior implicação para o profissional de educação é, no entanto, o uso do computador como ferramenta de ensino. O computador será usado para auxiliar o aluno a aprender, estudar e praticar exercícios sobre os mais variados assuntos, revolucionando totalmente os métodos de ensino.

A primeira reação do profissional é de temor já que o mito "computador gera desemprego" se tornou familiar. Não vemos porém a substituição do professor pelo computador, vemos o computador como um "auxiliar de ensino", como um "laboratório versátil" onde o aluno poderá se exercitar com interesse.

Para tanto, será necessário desenvolver programas de ensino, os educadores poderão fazer mais do que escrever livros ao participar da elaboração de tais programas; é um novo mercado de trabalho que se abre para os educadores.

1.5. REPRESENTAÇÃO DE DADOS

Computadores de todos os tamanhos têm uma coisa em comum, os dados. Em computadores digitais, os números são representados no sistema de numeração binário. A principal razão para utilizar dígitos binários em computadores é a simplicidade como dispositivos elétricos, magnéticos ou mecânicos podem representar dígitos binários.

Sistema Decimal

O sistema de numeração decimal é composto por 10 símbolos, algarismos ou dígitos: 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9. O sistema decimal é também chamado de sistema de numeração de base 10, porque 10 símbolos diferentes são utilizados.

No sistema decimal cada dígito assume um valor diferente em função da posição dentro do número.

Por exemplo:

$$\begin{array}{lllr} 123 & \Longrightarrow 1 & \longrightarrow & 100 \\ & 2 & \longrightarrow & 20 \\ & 3 & \longrightarrow & 3 \\ \hline & & & 123 \end{array}$$

As várias posições dos dígitos nos números podem ser expressas em potências de 10, o número 123 poderia ser representado por:

$$(1 \times 10^2) + (2 \times 10^1) + (3 \times 10^0).$$

Sistema Binário

No sistema binário apenas os dígitos 0 e 1 são utilizados, e o sistema é denominado sistema de base 2. Neste sistema é possível representar qualquer quantidade que possa ser representada no sistema decimal ou em outro sistema de numeração. Como o número de dígitos diferentes permitidos (2) é muito menor do que o número de dígitos possíveis no sistema decimal, a representação de uma mesma quantidade exigirá um número maior de dígitos no sistema binário para ser representada.

$123_{10} = 1111011_2$

$(1 \times 2^6) + (1 \times 2^5) + (1 \times 2^4)\ (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)$

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	→ valores de cada posição
↓	↓	↓	↓	↓	↓	↓	↓	
7	6	5	4	3	2	1	0	
0	1	1	1	1	0	1	1	

Conversão Binário para Decimal

A conversão de uma quantidade expressa por um número binário é facilmente convertida para sistema decimal, pelo conhecimento do valor de cada posição.

0		0		0		1		1		1		0		1	
↓		↓		↓		↓		↓		↓		↓		↓	
0	+	0	+	0	+	2^4	+	2^3	+	2^2	+	0	+	2^0	$= 29_{10}$

Conversão Decimal para Binário

A conversão decimal para binário pode ser obtida por divisões sucessivas por 2, até que se obtenha o quociente zero. Os restos obtidos formarão o número na base 2.

Complemento a Um

Em complemento a um, todos os números inteiros positivos são representados em seu formato correto, isto é, o número +6, é representado na forma usual: 00000110. Contudo, o seu complemento, −6, é obtido pela complementação de cada bit da representação original. Cada dígito 0 é transformado em 1, e cada dígito 1 é transformado em 0. Desta forma o número −6, será representado por 11111001.

Observe que a representação em complemento a um apresenta uma dificuldade, que é a existência de duas formas diferentes para representar o 0, esse problema é também denominado ambigüidade do zero. De fato se considerarmos o número 00000000, o seu complemento seria 11111111, que corresponderia a zero negativo, que não tem significado. Na prática esse zero negativo quando detectado é transformado em zero normal.

Complemento a Dois

Em complemento a dois, os números positivos são representados de forma usual, e os números negativos são representados de forma semelhante a representação em complemento a um. A diferença é que o complemento a dois é obtido pela aplicação da regra de complemento a um seguido da adição de um 1, como podemos ver no exemplo:

```
(+6) ——→ 00000110
(−6) ——→ 11111001   (complemento a um)
       +        1
         ————————
         11111010   (complemento a dois)
```

À primeira vista a representação com base no complemento é bastante complicada. Para melhor esclarecer o leitor, vamos explicar como seria usada esta representação na base 10.

Vamos supor que estamos lidando com números de 4 algarismos apenas; para cada nümero X (positivo) podemos encontrar outro número X' (também positivo) que funcione como uma representação conveniente para o número −X.

Basta observar que se $X + (-X) = 0$ deveremos também ter $X + X' = 0$ para que X' seja equivalente a −X. Mas como pode $X + X' = 0$ se X e X' são ambos positivos? Aí é que entra o número de algarismos; se estamos considerando apenas 4 algarismos, então $X + X' = 10000$ é o mesmo que $X + X' = 0000$ já que o 5º algarismo será perdido.

Como achar o valor de X'?

Se $X + X' = 10000$, então $X' = 10000 - X$

Porém existe um método mais fácil de se calcular $10000 - X$ (principalmente porque estamos trabalhando com apenas 4 algarismos):

Como todos sabem $10000 = 9999 + 1$; logo, $X' = 9999 - X + 1$

(Note-se também que a subtração de 9999 é mais simples porque não ocorre o "pede emprestado".)

Assim, mesmo na base 10 existe para cada número X um complemento a 10, X' que funciona como se fosse −X sendo também possível calculá-lo subtraindo X de 9999... (complemento a 9) e adicionando-se 1.

Exemplo: Calcular o complemento a 10 de 385:

$$X' = 9999 - 385 + 1 = 9614 + 1 = 9615$$

Pode-se verificar que, para qualquer operação com 4 algarismos, somar com 9615 é equivalente a subtrair 385.

Finalmente, na base 2, o processo é muito semelhante. Para cada número Y pode-se ter um número Y', seu complemento a 2 que funcione como −Y. Admitindo-se que estamos trabalhando agora com oito algarismos (bits), teremos

$$Y + Y' = 100000000$$

$$Y' = 100000000 - Y$$

$$Y' = 11111111 + 1 - Y$$

$$Y' = 11111111 - Y + 1$$

Porém, na base 2 subtrair um bit de 1 equivale a inverter este bit, já que $1 - 0 = 1$ e $1 - 1 = 0$ e o resultado $11111111 - Y$ será facilmente obtido invertendo-se os bits de Y. O resultado $\overline{Y}$ é o complemento a 1 de Y.

Por isso, abreviando-se o processo, se diz que o complemento a 2 é obtido trocando-se os bits e somando-se 1.

Números Positivos e Negativos

Quando desejamos representar números positivos e negativos reduzimos a magnitude máxima que se poderia obter. Por exemplo, considerando números de 8 dígitos binários poderíamos representar números entre 0 e 255, caso desejássemos representar também números negativos, com os mesmos 8 dígitos poderíamos representar números entre −128 e +127, que usualmente são representados em complemento a dois, desta forma, por inspeção do dígito de maior significado, podemos verificar se o número referenciado é positivo ou negativo (1, para números negativos, 0, para números positivos).

Representação de Dados Alfanuméricos

A representação de dados alfanuméricos, isto é, caracteres, é feita através de códigos previamente estabelecidos, isto é, em geral, todos os caracteres são representados em um código de oito dígitos binários.

Em geral, em computação são utilizados: códigos ASCII e códigos EBCDIC.

ASCII (American Standard Code for Information Interchange), é universalmente utilizado em microcomputadores. EBCDIC é uma variação do ASCII, utilizada principalmente pela IBM, e normalmente são utilizados em microcomputadores para interface com outros sistemas.

A codificação ASCII é formada por 26 letras, maiúsculas e minúsculas, 10 símbolos numéricos, e mais 20 símbolos especiais, conforme pode ser visto na Fig. 1.5. Estes códigos utilizam 7 dígitos binários, de tal forma que é possível representar 128 dígitos diferentes.

O oitavo dígito binário, quando usado, é utilizado com dígito de paridade.

DECIMAL	OCTAL	HEXADECIMAL	ASCII	DECIMAL	OCTAL	HEXADECIMAL	ASCII	DECIMAL	OCTAL	HEXADECIMAL	ASCII	DECIMAL	OCTAL	HEXADECIMAL	ASCII
0	000	00	NUL	32	040	20	SP	64	100	40	@	96	140	60	`
1	001	01	SOH	33	041	21	!	65	101	41	A	97	141	61	a
2	002	02	STX	34	042	22	"	66	102	42	B	98	142	62	b
3	003	03	ETX	35	043	23	#	67	103	43	C	99	143	63	c
4	004	04	EOT	36	044	24	$	68	104	44	D	100	144	64	d
5	005	05	ENQ	37	045	25	%	69	105	45	E	101	145	65	e
6	006	06	ACK	38	046	26	&	70	106	46	F	102	146	66	f
7	007	07	BEL	39	047	27	'	71	107	47	G	103	147	67	g
8	010	08	BS	40	050	28	(	72	110	48	H	104	150	68	h
9	011	09	HT	41	051	29	)	73	111	49	I	105	151	69	i
10	012	0A	LF	42	052	2A	*	74	112	4A	J	106	152	6A	j
11	013	0B	VT	43	053	2B	+	75	113	4B	K	107	153	6B	k
12	014	0C	FF	44	054	2C	,	76	114	4C	L	108	154	6C	l
13	015	0D	CR	45	055	2D	-	77	115	4D	M	109	155	6D	m
14	016	0E	SO	46	056	2E	.	78	116	4E	N	110	156	6E	n
15	017	0F	SI	47	057	2F	/	79	117	4F	O	111	157	6F	o
16	020	10	DLE	48	060	30	0	80	120	50	P	112	160	70	p
17	021	11	DC1	49	061	31	1	81	121	51	Q	113	161	71	q
18	022	12	DC2	50	062	32	2	82	122	52	R	114	162	72	r
19	023	13	DC3	51	063	33	3	83	123	53	S	115	163	73	s
20	024	14	DC4	52	064	34	4	84	124	54	T	116	164	74	t
21	025	15	NAK	53	065	35	5	85	125	55	U	117	165	75	u
22	026	16	SYN	54	066	36	6	86	126	56	V	118	166	76	v
23	027	17	ETB	55	067	37	7	87	127	57	W	119	167	77	w
24	030	18	CAN	56	070	38	8	88	130	58	X	120	170	78	x
25	031	19	EM	57	071	39	9	89	131	59	Y	121	171	79	y
26	032	1A	SUB	58	072	3A	:	90	132	5A	Z	122	172	7A	z
27	033	1B	ESC	59	073	3B	;	91	133	5B	[	123	173	7B	{
28	034	1C	FS	60	074	3C	<	92	134	5C	\	124	174	7C	|
29	035	1D	GS	61	075	3D	=	93	135	5D	]	125	175	7D	}
30	036	1E	RS	62	076	3E	>	94	136	5E	^	126	176	7E	~
31	037	1F	US	63	077	3F	?	95	137	5F	_	127	177	7F	DEL

Fig. 1.5 — Tabela de código ASCII.

1.6. EXEMPLO DE APLICAÇÃO

Uma aplicação bastante didática do uso de microprocessadores seria o controle de um sinal de trânsito para travessia de pedestres conforme ilustrado na Fig. 1.6.

O funcionamento deste dispositivo é já bastante conhecido: temos um sinal com duas lâmpadas (vermelha e verde) e um botão que pode ser acionado pelo pedestre. Normalmente, a lâmpada verde está acesa e a vermelha apagada

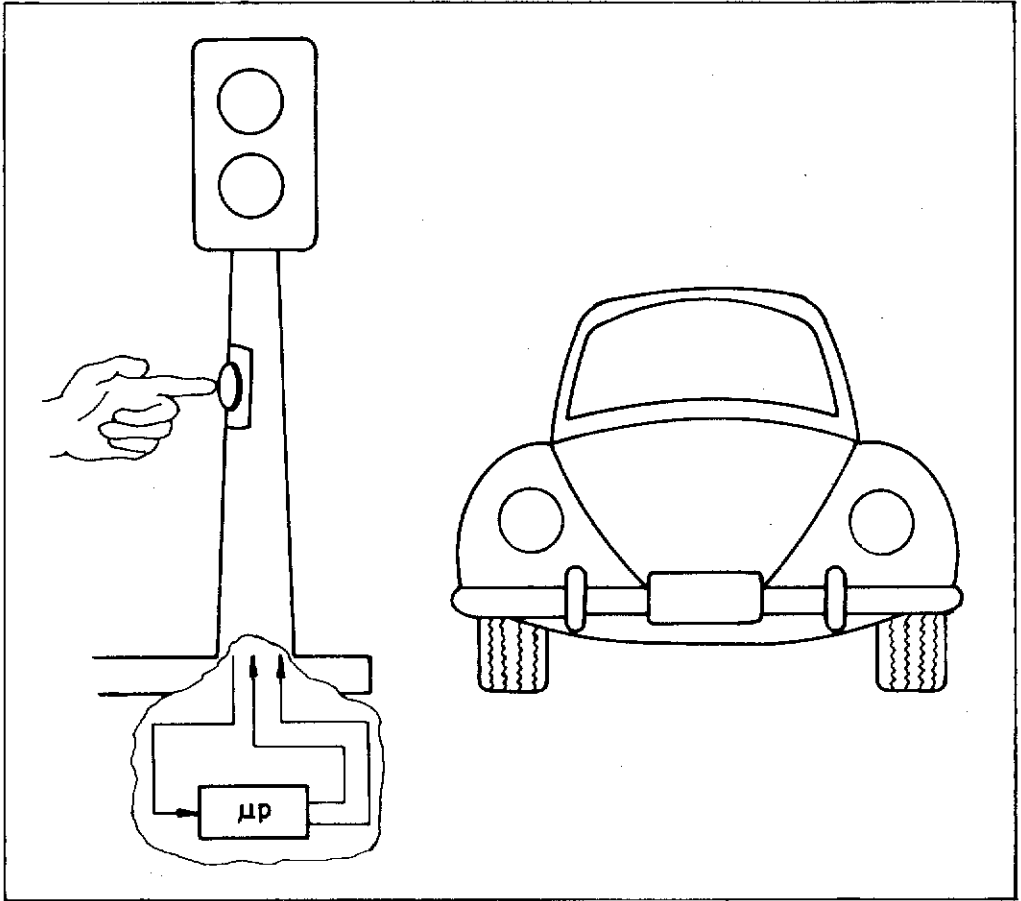



Fig. 1.6 — Sinal de pedestre.

permitindo o tráfego de automóveis. Quando um pedestre aciona o botão, o sinal passa por uma transição mantendo ambas as luzes acesas e, após um período de 10 segundos, apaga a luz verde mantendo apenas a vermelha acesa. Passados 30 segundos, o sinal reabre, retomando o estado inicial.

O circuito que controla o sinal recebe uma informação (se o botão está apertado ou não) e decide o que fazer com dois fios: um que pode ligar à lâmpada verde e outro à vermelha.

No caso de termos um controle por microprocessador, todas as decisões são comandadas pelo programa que se encontra no processador. A Fig. 1.7 mostra o fluxograma de um programa simples.

Este mesmo exemplo será estudado e resolvido com detalhes no Cap. 4. No momento vamos aproveitá-lo para esclarecer alguns conceitos:

Armazenamento do Programa

Em um computador comum o programa é carregado na memória através de leitoras de cartões, fitas, discos etc. Isto, obviamente não seria prático neste

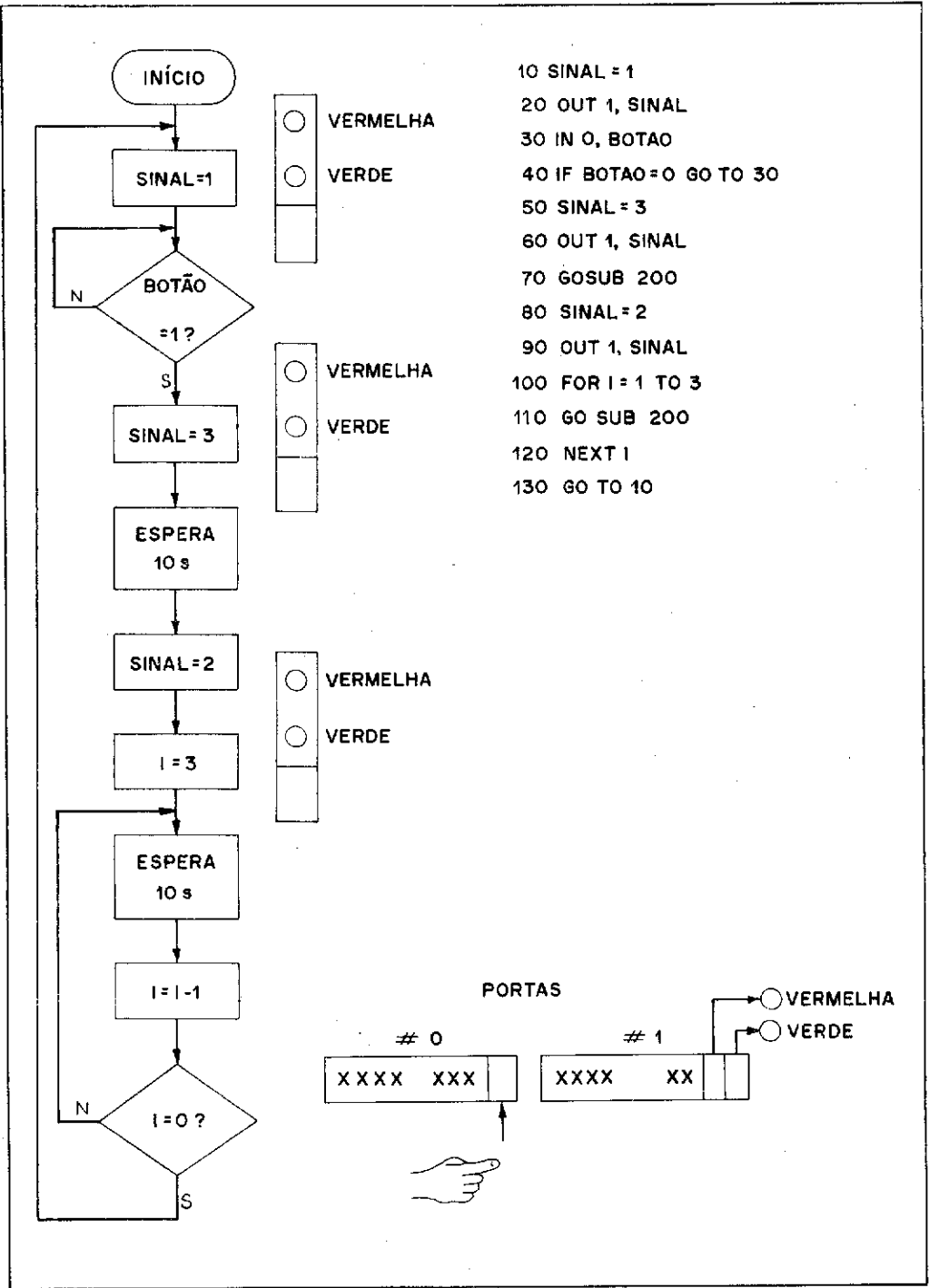



Fig. 1.7 — Sinal de Pedestre: Software.

caso. Ao invés disso, usa-se um tipo especial de memória (discutida no Cap. 2) na qual o programa é gravado permanentemente.

Entrada e Saída

Em qualquer computador existem informações que entram para serem processadas e informações que são produzidas neste processamento e então comunicadas ao mundo exterior. Essas informações de Entrada e Saída são normalmente registradas através de cartões perfurados, fitas ou discos magnéticos, papel impresso, terminais de vídeo etc.

Em nosso texto lidaremos com informações de forma mais básica: todas as informações são sinais elétricos tais como ilustra o exemplo.

Sinais Analógicos

No exemplo estudado, os sinais de Entrada e Saída eram digitais e binários; binários porque só assumem dois estados (um/zero; ligado/desligado) e digitais porque não existe uma variação contínua de um estado para outro.

Em alguns problemas pode ser necessário lidar com grandezas analógicas tais como temperatura, velocidade etc. Essas grandezas são convertidas para sinais elétricos (tal como uma corrente elétrica que varie conforme a temperatura) porém o computador não pode lidar diretamente com sinais que variam continuamente, sendo necessário ainda efetuar uma conversão do sinal analógico para um conjunto de sinais digitais que expressarão um número proporcional ao valor analógico (Fig. 1.8).

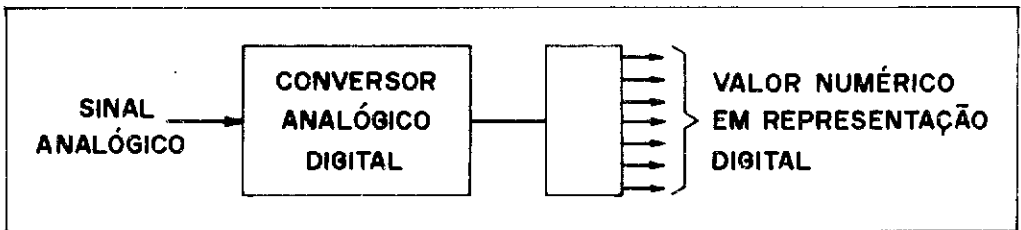



Fig. 1.8 — Conversor analógico digital.

1.7. EXERCÍCIOS

1. Qual o valor decimal de 101101_2?
2. Converta 57_{10} para binário?
3. O sistema hexadecimal, ou base 16, se utiliza de 16 algarismos diferentes, represente na base 16 os seguintes valores:

a) 31_{10}
b) 01011101_2

4. Represente em complemento a nove e complemento a dez os seguintes números:

a) − 24
b) − 7462

5. Represente em complemento a um e complemento a dois os seguintes números:

a) 47_{10}
b) $3F_{16}$
c) -31_{10}
d) -107_{10}

6. Mostre como somar em complemento a um (com 6 bits), os seguintes números:

a) 8_{10} e -11_{10}
b) -7_{10} e $2A_{16}$

7. Repita o exercício anterior para complemento a dois.

8. O número 123.56 em ponto flutuante pode ser representado por:

$$1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 + 5 \times 10^{-1} + 6 \times 10^{-2},$$

sendo também válida a equação $N = m \times b^e$, onde:

N = número em ponto flutuante
m = mantissa, que definirá a precisão em algarismos significativos
b = base
e = expoente
$N = .12356 \times 10^3$

Mostre um possível formato para representar números, um ponto flutuante utilizando 16 bits, e represente os seguintes números:

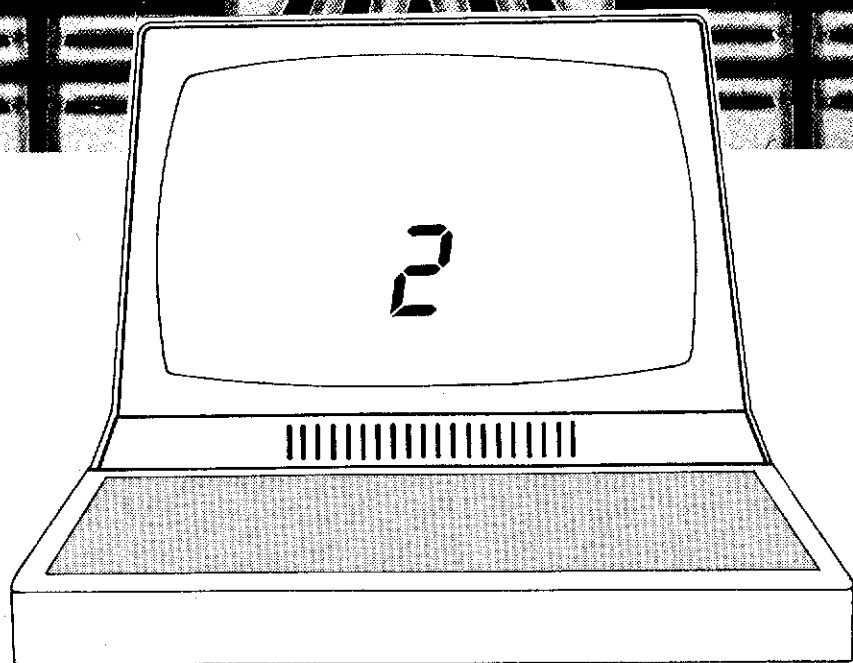
a) 123.21_{10}
b) $.074_{10}$
c) 34.31_{10}

9. No código BCD (Binary Coded Decimal), cada dígito decimal é representado por 4 bits (0000(0) à 1001(9)), e portanto um byte pode armazenar dois dígitos decimais, isto é, valores no intervalo 0 a 99. Utilizando dois bytes (16 bits), represente em BCD os seguintes números:

a) 921_{10}
b) 37_{10}
c) $23F_{16}$

10. Explique como poderemos representar números negativos em BCD.

11. Represente em binários os códigos ASCII dos seguintes caracteres:

a) A
b) 7
c) CR (Carriage return)

ARQUITETURA DE COMPUTADORES

2.1. CIRCUITOS DIGITAIS

Os circuitos normalmente utilizados na construção de computadores reconhecem dois tipos de estados, denominados estado 0 e estado 1. Pelo fato de serem dois estados, estes circuitos são também chamados de binários.

A implementação destes circuitos é feita através de circuitos elétricos e, assim sendo, os dois estados (0 e 1) são reconhecidos conforme haja ou não eletricidade em determinado ponto do circuito.

Ainda assim, haver ou não haver eletricidade é muito vago pois poderia ocorrer que tivéssemos muito pouca eletricidade e pensarmos que não houvesse nenhuma. Nos últimos anos, grande parte dos fabricantes tem seguido uma padronização coerente, conhecida como padrão TTL (lógica Transistor-Transistor). De acordo com este padrão devemos reconhecer como estando no estado 0 uma linha cujo nível de tensão esteja entre −0,5V e 0,8V e como estando no estado 1 uma linha cujo nível de tensão esteja entre +2V e +5,5V. Note-se, neste caso, a existência de um grande intervalo (+0,8V a +2V) que seria uma espécie de "Zona Indefinição", a fim de permitir uma identificação clara do estado desejado.

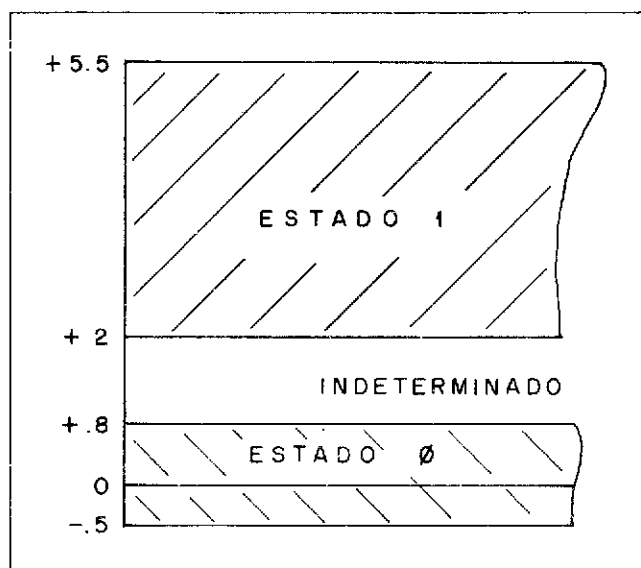



Fig. 2.1 — Níveis lógicos TTL.

Inversor (NOT)

O primeiro elemento de circuito lógico que vamos estudar é o inversor, representado conforme a Fig. 2.2. O inversor, como o nome indica, inverte o

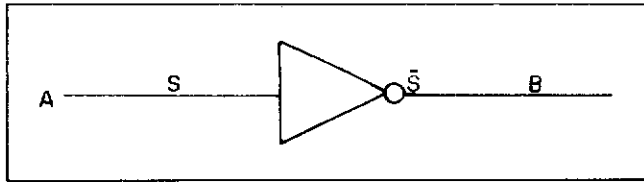



Fig. 2.2 — Inversor (NOT).

sinal recebido; isto é, se tivermos na entrada (A) um "1" teremos na saída (B) um "0" e vice-versa. Outro modo de representar sua ação é dizermos que se temos na entrada um sinal S teremos na saída $\overline{S}$.

"E" Lógico (AND)

Outro elemento importante é a porta denominada "E". Na Fig. 2.3 ilustramos uma porta "E" com duas entradas (A e B). A saída C será igual a "1" se A "E" B forem iguais a "1" (daí o nome da porta). Caso contrário, isto é, se A ou B (ou ambos) forem "0", a saída será "0".

Existem portas "E" com mais de duas entradas, sendo a saída igual a "1" se todas as entradas forem "1".

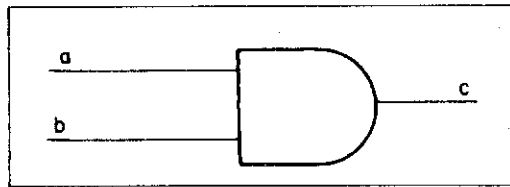



Fig. 2.3 — "E" lógico (AND).

"OU" Lógico (OR)

A porta "OU" com duas entradas é ilustrada na Fig. 2.4. Sua saída C será igual a "1" se A "OU" B (ou ambos) forem iguais a "1". Será zero em caso contrário (isto é, quando A e B forem ambos "0").

Existem portas "OU" com mais de duas entradas sendo a saída igual a "1" se pelo menos uma das entradas for "1".

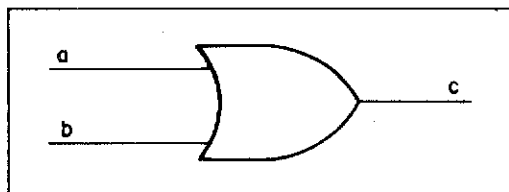



Fig. 2.4 — "OU" lógico (OR).

Outras Portas Lógicas

Com as portas "OU", "E" e inversores pode-se implantar qualquer função lógica. Existem outros tipos de portas que poderão ser mais convenientes dependendo da situação.

A porta "OU-INVERTIDO" (NOR) funciona exatamente como um "OU" seguido de um inversor como mostra a Fig. 2.5(*a*) e é representada abreviadamente como a Fig. 2.5(*b*). O leitor poderá verificar, como exercício, que as Figs. 2.5(*b*) e 2.5(*c*) são equivalentes.

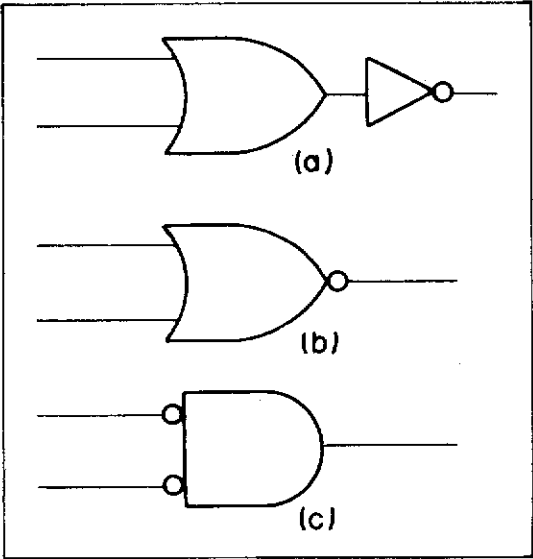



Fig. 2.5 — "OU-invertido" (NOR).

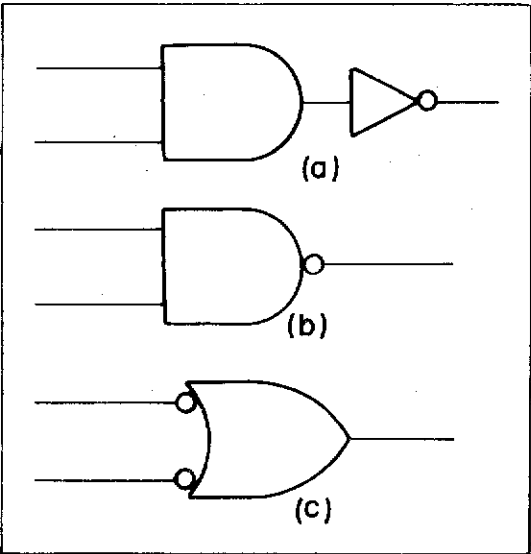



Fig. 2.6 — "E-invertido" (AND).

A porta "E-INVERTIDO" (NAND) funciona como um "E" seguido de um inversor como mostra a Fig. 2.6(*a*), e é representada como mostra a Fig. 2.6(*b*). O leitor poderá verificar, como exercício, que as Figs. 2.6(*b*) e 2.6(*c*) são equivalentes.

Circuito Hidráulico

Um circuito de computador poderia ser implementado usando-se outros princípios além da eletricidade. Um exemplo conveniente para entender como funcionam os circuitos lógicos é o da implementação hidráulica.

Na Fig. 2.7 temos um reservatório de água ligado a um encanamento através de um registro R_1. Quando R_1 está aberto a água da caixa entra no encanamento, e quando o mesmo estiver bem cheio começará a sair água pelo chuveiro que se encontra na outra extremidade. Assim, o indivíduo A pode colocar o encanamento no estado lógico "1" abrindo o registro R_1 e fechando R_2 ou pode colocá-lo no estado lógico "0" fechando R_1 e abrindo R_2. O indivíduo da extremidade B poderá saber se o estado é o 0 ou 1 pela presença ou ausência de água no chuveiro.

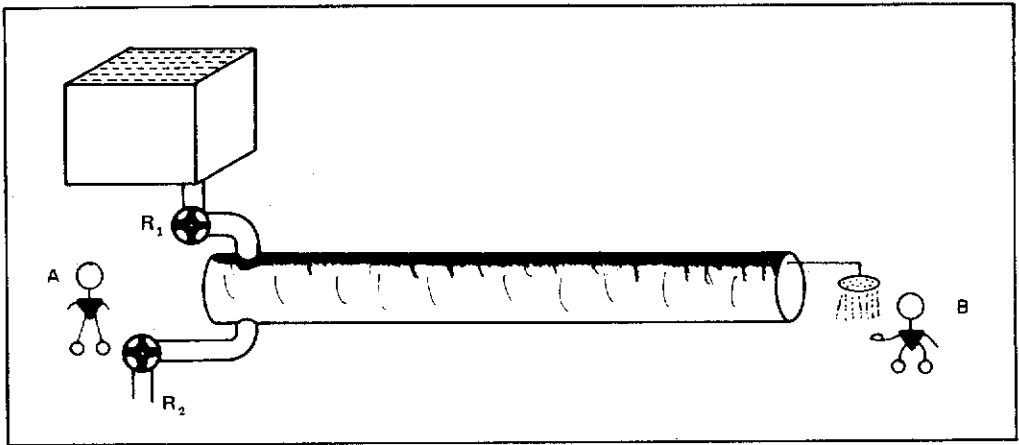



Fig. 2.7 — Circuito hidráulico.

O funcionamento de um inversor hidráulico poderia ser ilustrado pela Fig. 2.8. O indivíduo que se encontra dentro do inversor é instruído de modo a inverter o sinal observado na entrada; se a entrada for "0" (chuveiro desligado) ele abre R_1 e fecha R_2 o que causa o encanamento de saída encher, indo para o estado 1. Se a entrada for "1" (chuveiro ligado) ele fecha R_1 e abre R_2 esvaziando o encanamento de saída que vai para o estado lógico "0".

Circuitos Lógicos e as Realidades da Vida

Tempo de Propagação. Os circuitos não reagem imediatamente a uma mudança; por exemplo, se tivéssemos na entrada de um inversor o estado lógico

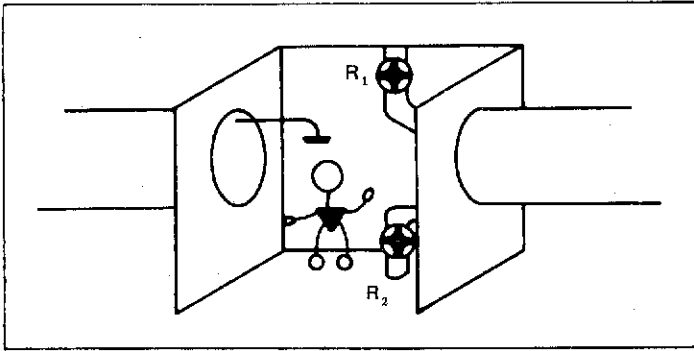



Fig. 2.8 — Inversor hidráulico.

"0" e mudássemos para "1", a saída que estava em "1" deveria mudar para "0"; porém isto não ocorre instantaneamente. Recorrendo ao circuito hidráulico, vemos que o indivíduo responsável deverá:

1. observar a mudança na entrada; ou seja, que começou a cair água;
2. inverter o estado da saída; ou seja, fechar R_1 e abrir R_2.

Além disso, o encanamento de saída, que estava cheio, levará algum tempo até esvaziar.

O tempo total transcorrido entre uma mudança na entrada e a correspondente mudança na saída é chamado o tempo de propagação ou tempo de retardo (DELAY).

Apesar de que um circuito eletrônico tenha transições muito mais rápidas do que o circuito hidráulico do exemplo, esses tempos existem e não podem ser desprezados. Existem circuitos fabricados de diversas formas, o que implica, dentre outras coisas, em diferentes tempos de propagação. Uma das tecnologias mais comuns, usada na família TTL tem tempo de propagação da ordem de 10 nanossegundos (10 ns), ou seja, 0,00000001 segundo.

Fan-out

A saída de uma porta pode alimentar diversas outras conforme mostra a Fig. 2.9. O inversor A alimenta a entrada de três outras portas. No entanto, não

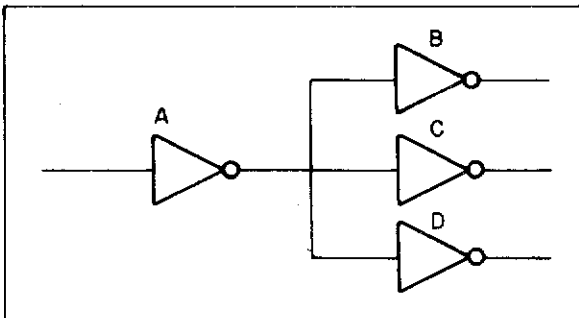



Fig. 2.9 — Fan-Out.

podemos aumentar indefinidamente este número porque cada porta adicional irá consumir energia chegando a um ponto onde a porta A excederá sua capacidade e poderá falhar.

O problema pode ser facilmente percebido no circuito hidráulico (Fig. 2.10): à medida que colocamos mais extensões, o maior número de chuveiros vai consumir mais água podendo ocorrer que a alimentação que vem da caixa d'água não seja suficiente para manter todos os chuveiros; ocorrerá então que alguns terão água e outros não, o que é uma condição de erro.

O número máximo de "extensões" que se pode tirar da saída de uma porta é o que se chama "Fan-out". Para a família TTL esse número é da ordem de 10.

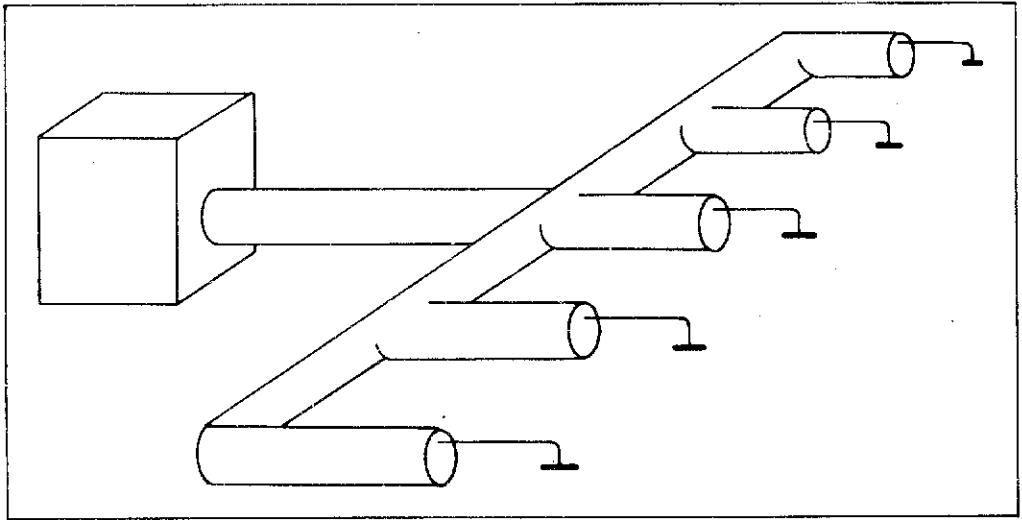



Fig. 2.10 — Fan-out hidráulico.

Saída de Coletor Aberto

Que acontece se juntamos as saídas de mais de uma porta lógica?

Se os sinais forem iguais (na Fig. 2.11: a = b) é de se esperar que nenhum problema ocorra; porém se forem diferentes (por exemplo, $\bar{a} = 1$, $\bar{b} = 0$) então teremos um conflito pois o inversor A tentará forçar sua saída para "1" ao passo

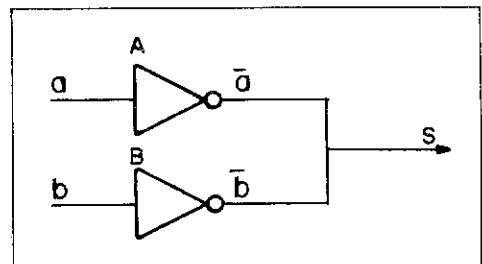



Fig. 2.11 — Conflito lógico.

que B tentará forçá-la para "0". Isto causará, além de uma condição de erro, dano aos inversores.

No circuito hidráulico essa condição seria fácil de explicar: A tentaria manter o encanamento cheio abrindo o registro que vem da caixa; B tentaria esvaziá-lo abrindo a saída para o ralo.

As portas lógicas usuais não podem, portanto, ter suas saídas ligadas. Existem dois tipos de portas lógicas cujas saídas podem ser ligadas: as portas de coletor aberto e as portas de três estados.

As portas com saída de coletor aberto seriam hidraulicamente equivalentes a um elemento cujo encanamento de saída (para o ralo) tivesse grande capacidade de vazão. Como podemos ver na Fig. 2.12, o encanamento de entrada é muito menor que o de saída; assim, se A, B e C, estiverem todos no estado lógico "1", as saídas de água estando fechadas e as entradas abertas, o encanamento ficará cheio, ou seja, no estado lógico "1". Se, porém, qualquer elemento A, B ou C, mudar sua saída para "0" ele esvaziará o encanamento já que as entradas não terão capacidade de mantê-lo cheio.

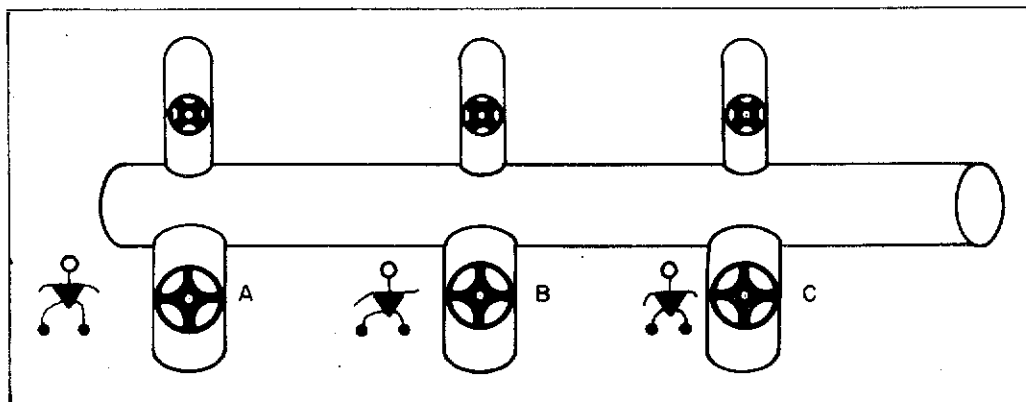



Fig. 2.12 — Saída em coletor aberto (hidráulica).

Este é, em resumo, o modo de funcionamento das portas com saídas em coletor aberto; se todas as portas estiverem com saída em "1", a saída será, efetivamente, "1"; se pelo menos uma porta estiver com a saída em "0", a saída será, efetivamente, "0". Como este resultado é semelhante à operação de uma porta "E" lógica, é comum denominar-se um conjunto de portas coletor aberto pelo nome de "E fiado" (Wired-And) pois neste caso o efeito "E" é obtido por fiação e não por intervenção de mais uma porta lógica.

As portas com saídas de coletor aberto serão representadas pelos símbolos normais acompanhados de um asterisco (Fig. 2.13). Essas portas devem ter sua saída ligada à fonte de alimentação (+5V) através de uma resistência calculada em função do número de portas ligadas.

Saídas de Três Estados

Outro tipo importante de portas lógicas largamente utilizado com microprocessadores são as portas com saídas em três estados. Como o nome procura

dizer, ao invés de 2 possíveis estados "0" ou "1", existe um terceiro estado, intermediário, chamado indeterminado ou estado de "alta impedância".

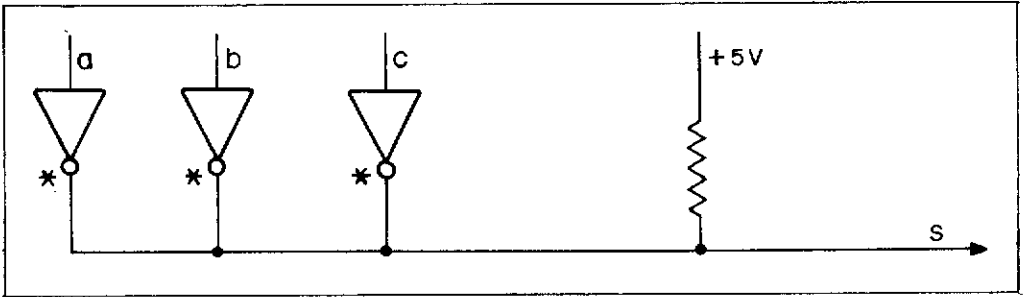



Fig. 2.13 — Saída em coletor aberto.

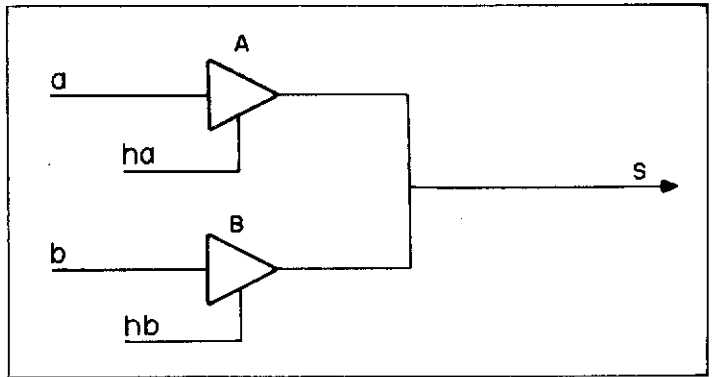



Fig. 2.14 — Saída de três estados.

As portas desse tipo têm, além do sinal de entrada, um sinal de controle de "habilitação" ("enable"). Na Fig. 2.14 temos duas portas A e B com sinais de entrada a e b e com sinais de habilitação ha e hb.

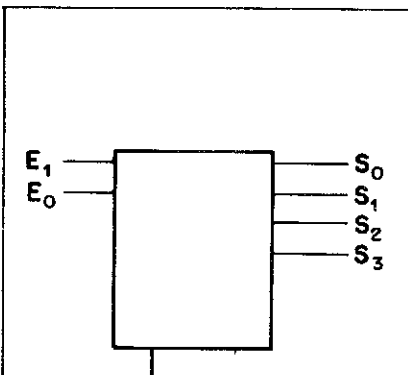
Cada porta opera do seguinte modo: se a habilitação está ligada, isto é, em "1" então a saída da porta é igual ao sinal de entrada. Se a habilitação está desligada, isto é, em "0", então a saída desta porta fica em estado indeterminado, independente do sinal de entrada.

No circuito hidráulico, a habilitação ligada seria uma operação normal; a habilitação desligada faria com que tanto o registro de entrada como o de saída fossem fechados. Isto é, não se tentaria colocar a saída nem em "0" nem em "1"; é como se esta porta estivesse fora do circuito. O valor da saída então seria determinado por alguma outra porta que estivesse habilitada (ou permaneceria indeterminado se nenhuma porta estivesse ligada).

É importante ressaltar que não se deve ter mais de uma porta habilitada em determinado momento.

2.2. INTEGRAÇÃO E OS BLOCOS PRÉ-FABRICADOS

A tecnologia eletrônica percorreu um longo caminho desde quando surgiram os primeiros computadores. Os primeiros circuitos integrados consistiam de alguns elementos encapsulados numa embalagem.

Tais circuitos já economizavam muito mais espaço e energia do que transistores discretos usados anteriormente; porém ainda eram como tijolos para fabricar uma casa. Um grande passo resultou da percepção que se poderia usar elementos pré-fabricados da mesma forma que paredes, portas e janelas na construção de casas. O estado da tecnologia já permitia colocar muitos circuitos numa mesma embalagem.

Dentre os circuitos mais populares temos os decodificadores e os registros.

Decodificador

O decodificador é um elemento de circuito capaz de selecionar uma dentre várias linhas de saída, de acordo com um código que lhe seja dado.

Na Fig. 2.15, ilustramos um decodificador de quatro linhas (S_0, S_1, S_2 e S_3). O circuito deverá ligar apenas uma dentre as quatro linhas de saída de acordo com o código representado pelas entradas E_1 e E_0. Assim, se o código na entrada for "0" ($E_1 = 0$ e $E_0 = 0$), então $S_0 = 1$, se o código for "1" ($E_1 = 0$, $E_0 = 1$), então $S_1 = 1$ e assim por diante.

Os decodificadores usualmente têm, ainda, uma entrada de habilitação h. Se a habilitação estiver desligada, nenhuma das saídas será selecionada (serão todas "0"). A tabela ilustra para cada possível valor das entradas h, E_1 e E_0, quais os valores das saídas S_0, S_1, S_2 e S_3.

ENTRADAS			SAÍDAS			
h	E_1	E_0	S_0	S_1	S_2	S_3
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

Fig. 2.15 — Funcionamento de um decodificador.

Os decodificadores podem ser variados sendo comuns decodificadores de 8 linhas e com mais de uma habilitação. Alguns decodificadores têm a saída invertida (a linha selecionada fica em "0" e as demais em "1").

Exemplo de projeto de um decodificador. Vamos imaginar que temos duas linhas E_1 e E_0 para indicar um código e vamos desprezar a habilitação. Ora, a saída S_0 deve ser "1" no caso de termos $E_1 = 0$ e $E_0 = 0$. Isto pode ser resolvido pelo circuito da Fig. 2.16(*a*).

Vejamos como ele funciona: S_0 está na saída de um "E" portanto só poderá ser "1" quando as duas entradas deste "E" forem 1. Estas entradas, por sua vez, são saídas dos inversores, cada uma delas será "1" se a respectiva entrada no inversor for "0"; portanto, somente no caso em que $E_1 = 0$ e $E_0 = 0$ teremos $S_0 = 1$ conforme desejamos.

Um outro modo de interpretar o circuito é observar que S_0 é "1" quando E_1 *não* for "1" "E" E_0 não for "1".

A linha S_1 deverá ser "1" quando o código 01 ($E_1 = 0$, $E_0 = 1$) aparecer; ou seja, S_1 é "1" quando E_0 for "1" "E" E_1 não for "1". Conforme está representado na Fig. 2.16(*b*).

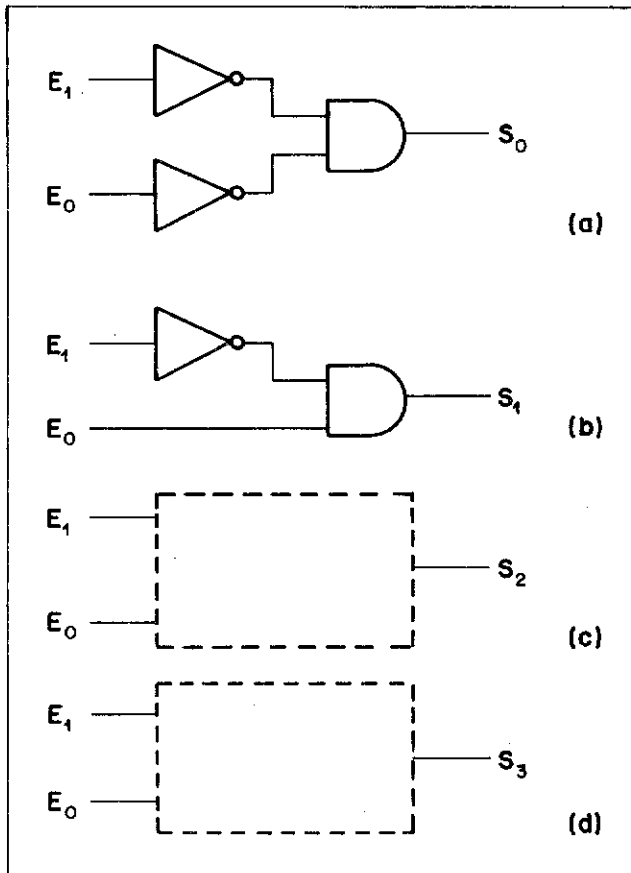



Fig. 2.16 — Construção de um decodificador.

O circuito para ativar as linhas S_2 e S_3 poderão ser completados pelo leitor a título de exercício.

Em nossos exemplos, estamos desenvolvendo os circuitos usando métodos intuitivos porém, o leitor interessado poderá consultar a literatura especializada para conhecer métodos mais diretos.

Na Fig. 2.17(*a*), temos apenas uma representação equivalente porém mais compacta do que a anterior. Repare que só temos necessidade de dois inversores para termos os sinais E_0 e E_1 invertidos.

A habilitação poderia ser implantada facilmente com mais uma porta "E". Finalmente o circuito seria encapsulado numa embalagem conforme indicado na Fig. 2.17(*b*).

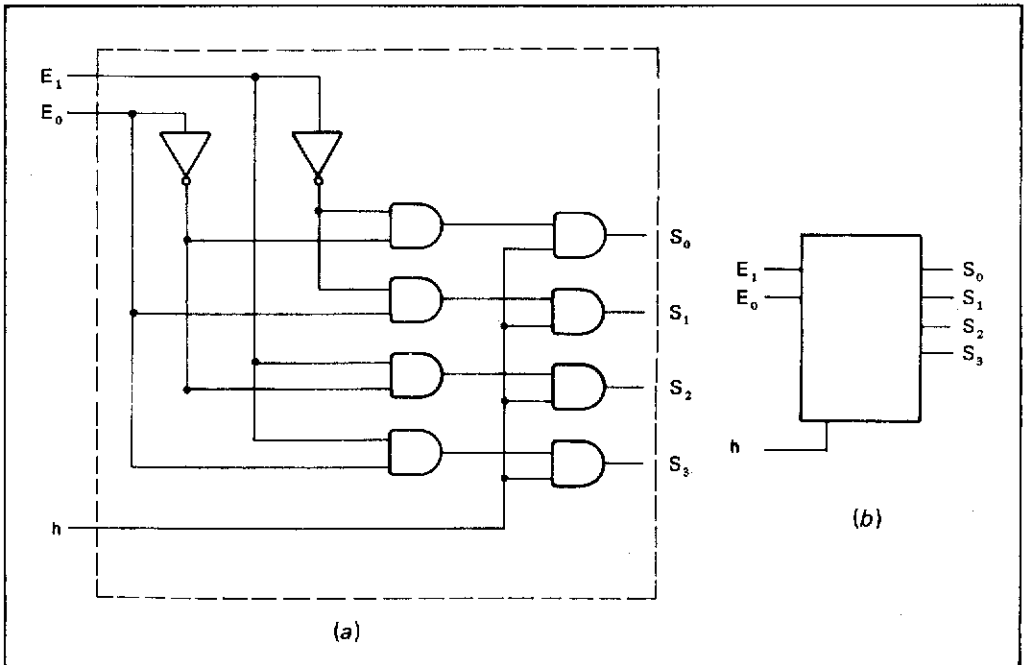



Fig. 2.17 — Decodificador.

Somadores

O somador é um circuito capaz de efetuar a soma de dois números binários (Fig. 2.17.1).

Na Fig. 2.17.1 ilustramos um somador de 4 bits onde os números A e B (de 4 bits cada um) são somados dando um resultado S (também de 4 bits) e um indicador de "vai um". Ao lado vemos dois exemplos de operação:

No primeiro temos uma soma de 5 com 4 dando resultado 9 sem "vai um". No segundo temos uma soma de 7 com 10 dando resultado 1 e causando "vai um" já que a soma (17 = 1 0001) não cabe em 4 bits.

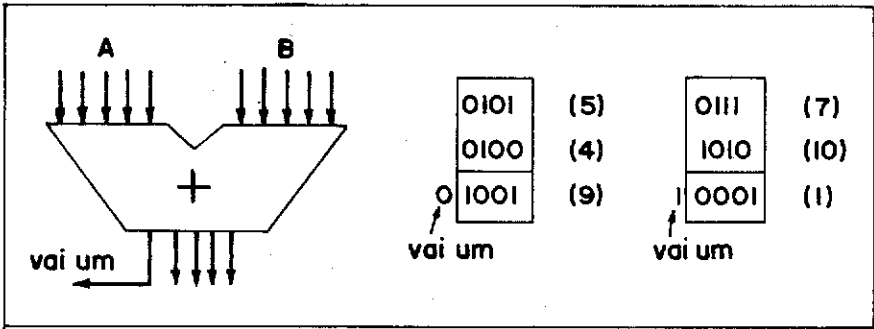



Fig. 2.17.1 — Somador de 4 bits.

Como o leitor já pôde constatar, a "tabuada" binária é muito mais simples que a decimal:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 e vai um

O circuito para cálculo da soma de dois bits (sem considerar "vai um") é muito simples de ser implementado usando blocos OU-EXCLUSIVO ou AND, OR, NOT como mostra a Fig. 2.17.2. Este circuito é chamado "meio-somador" (half-adder). Um somador completo (full-adder) deve levar em conta o "vai um" de uma casa anterior (Ce) e gerar o "vai um" da próxima casa (Cs) como mostra a Fig. 2.17.3(*b*).

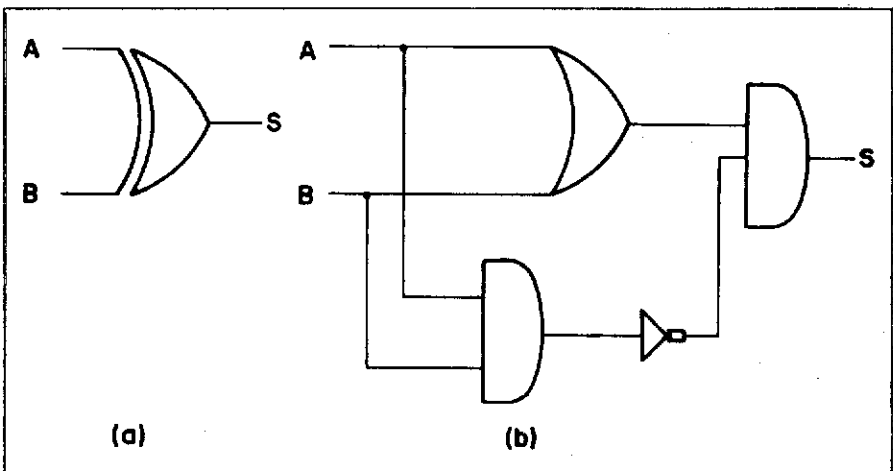



Fig. 2.17.2 — OU-EXCLUSIVO.

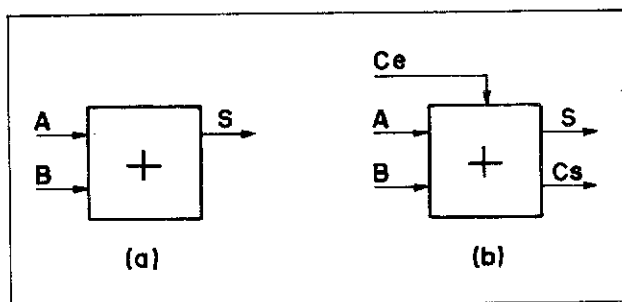



Fig. 2.17.3 — Meio-somador (*a*) e somador completo (*b*).

Um somador completo pode ser facilmente construído usando-se dois "meios-somadores". Um somador de vários bits como o da Fig. 2.17.1 pode ser facilmente construído usando-se vários somadores completos (ver exercícios do Cap. 2).

Unidade Aritmética

A unidade aritmética (Fig. 2.17.4) é um circuito capaz de realizar várias operações aritméticas e não somente a soma como no somador. Além dos dois operandos A e B são necessários alguns bits de controle que vão informar o tipo de operação desejada (soma, subtração, multiplicação etc.). Além do resultado R a unidade fornece também indicadores do resultado (vai um, resultado igual a zero etc.).

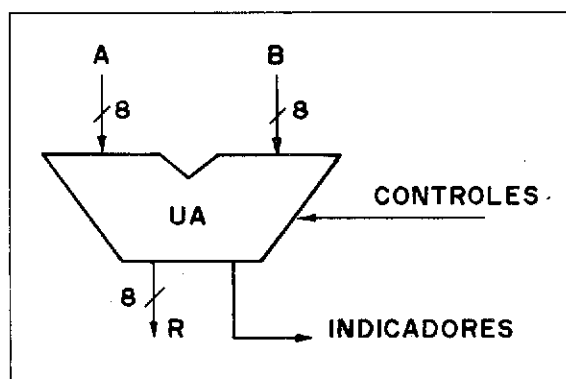



Fig. 2.17.4 — Unidade aritmética de 8 bits.

Registros de Depósito de Dados

Pode-se dizer que registros e vias de comunicação constituem o assunto fundamental da arquitetura de computadores. Da quantidade de registros e do

quão eficientemente os mesmos estão interligados dependerá a eficiência do sistema. Assim sendo, é de fundamental importância um perfeito entendimento de como funcionam tais registros e de como se processa a comunicação de dados entre um registro e outro.

Nas seções seguintes vamos discutir o funcionamento dos registros de três estados e a comunicação de dados entre os mesmos.

Registros de Três Estados

O tipo de registro que vamos discutir pode receber o nome de "depósito" (LATCH) porque tem a capacidade de armazenar (ou trancar) os dados que lhe são apresentados.

A Fig. 2.18 mostra duas representações equivalentes de um registro de 8 bits. Na primeira [2.18(*a*)] todas as 8 linhas de entrada e todas as oito de saída estão representadas individualmente. Na segunda, [2.18(*b*)] cada conjunto de oito linhas foi representado, abreviadamente, por uma só linha acompanhada da indicação de sua largura, isto é, 8 bits.

Além das linhas de entrada e saída têm-se duas linhas de controle de muita importância: a habilitação de entrada (enable IN) e a habilitação de saída (enable OUT).

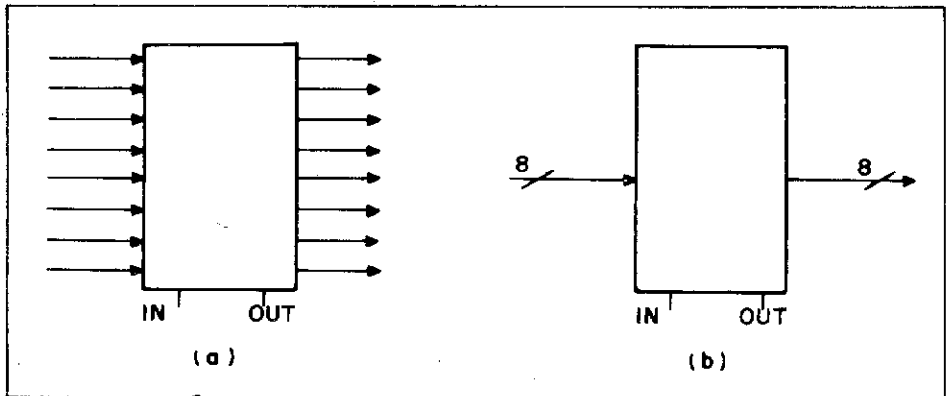



Fig. 2.18 — Registros de 3 estados.

Habilitação de Entrada (enable IN)

A habilitação de entrada é que comanda que os dados existentes nas linhas de entrada sejam copiados e armazenados em flip-flops no interior do registro. Dessa forma, enquanto IN = 0 o conteúdo do registro permanece inalterado a despeito do que possa ocorrer com as linhas de entrada (ou seja, o caminho entre cada linha e o interior do registro está interrompido). Quando IN = 1, o conteúdo do registro se identifica com as linhas de entrada (ou seja, o caminho é reestabelecido e o conteúdo da linha de entrada chega ao interior do registro).

Pode ocorrer, ainda, o caso de que o estado das linhas de entrada mude enquanto IN = 1. Apesar de que o resultado possa depender de características

do registro, consideraremos sempre o caso mais comum: se IN = 1 o conteúdo é igual à entrada, mesmo que esta varie; quando IN muda de 1 para 0, o último valor do registro (imediatamente antes da transição de IN) permanece armazenado no registro.

Habilitação de Saída (enable OUT)

A habilitação de saída é que permite que os dados armazenados no registro se propaguem pelas linhas de saída. Se OUT = 1 então as linhas de saída terão o mesmo valor que o conteúdo do registro; se OUT = 0 as linhas de saída são, virtualmente, desligadas do registro e seu valor não mais depende do conteúdo do registro. Pode-se então dizer que o estado dessas linhas com relação ao conteúdo do registro é INDETERMINADO. Note-se que esta indeterminação é apenas com relação ao registro em questão, que não fará qualquer tentativa de manter a saída sejam em 0 ou em 1.

Resumo: Sendo E o conteúdo das linhas de entrada, D o conteúdo do registro e S o conteúdo das linhas de saída, temos:

IN = 1 ⟹ D ← E
IN = 0 ⟹ D permanece inalterado
OUT = 1 ⟹ S ← D
OUT = 0 ⟹ S ← ? (indeterminado)

Aplicação de Portas de Três Estados

Os registros podem ser ligados de modo a que possamos transferir dados entre eles. Os registros com saída de três estados apresentam a grande conveniência de que suas saídas podem todas ser interligadas sem gerar conflitos.

Nos registros tradicionais onde a saída só pode ser 0 ou 1 é impossível uma ligação do tipo que mostra a Fig. 2.19(*a*), pois que aconteceria se uma saída estivesse em 0 e outra em 1?

As duas portas lógicas entrariam em conflito até que uma delas, ou ambas, ficassem danificadas. Portas de três estados, como mostra a Fig. 2.19(*b*), podem ter suas saídas ligadas, pois cada porta somente tenta interferir no resultado quando sua habilitação (OUT) for 1. A habilitação funciona como um sinal devendo impedir que mais de uma porta tente usar a saída ao mesmo tempo. Obviamente se ocorrer o contrário, o mesmo conflito existirá.

Transferência entre Registros

Consideremos o circuito da Fig. 2.20, em que quatro registros têm tanto suas linhas de entrada como as de saída conectadas:

Suponhamos ser nossa intenção copiar para R_3 o conteúdo do R_0. Como proceder?

Ora, a solução é bastante simples e consistirá de se enviar o conteúdo de R_0 através das vias de conexão (B) e recolhê-lo em R_3. Para isso é necessário:

1. Habilitar a saída de R_0 de modo a que o conteúdo de R_0 possa se propagar através das linhas de conexão (B).

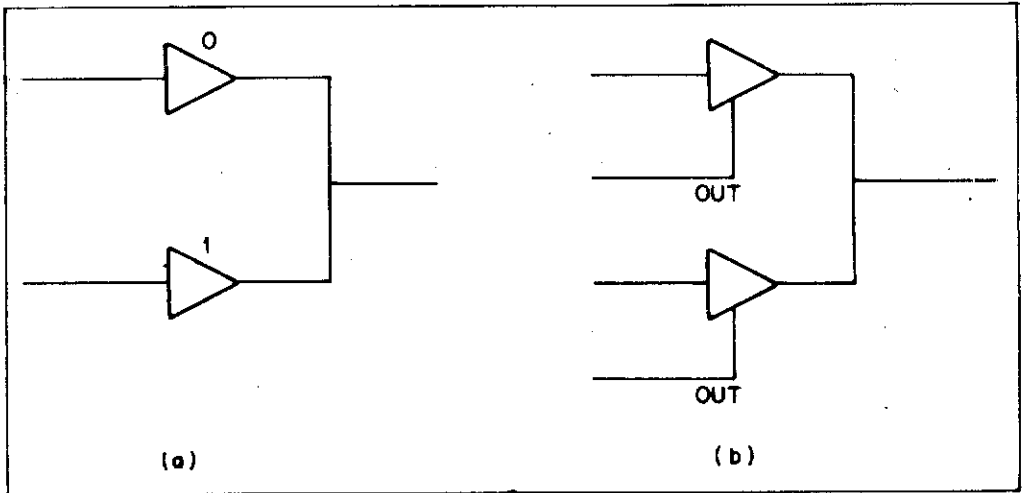



Fig. 2.19 — Conexão de saídas.

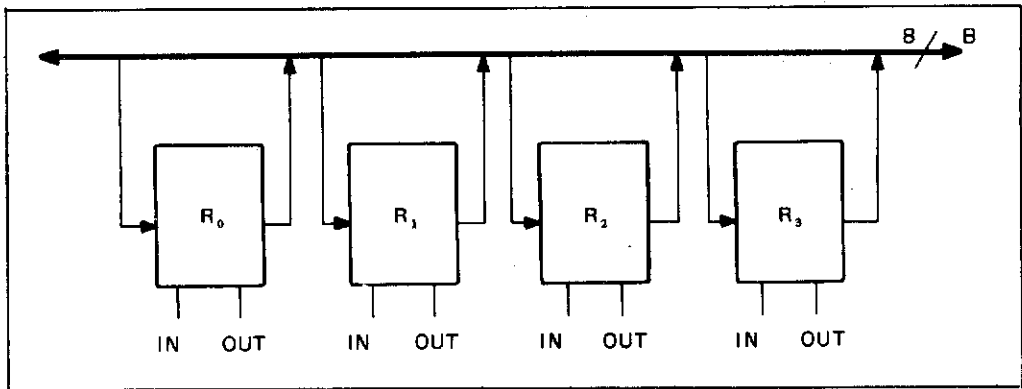



Fig. 2.20 — Transferência entre registros.

2. Habilitar a entrada de R_3 de modo a que as informações que estão nas vias de conexão (B) e que, portanto, se apresentam nas linhas de entrada, possam ingressar em R_3.

É importante ressaltar os seguintes fatos:

A informação que sai de R_0 irá fluir por todos os fios possíveis; porém, como só a entrada de R_3 estará habilitada, os dados entrarão em R_3.

A informação consome tempo para se propagar através de todos os fios e, por isso, os sinais de controle ($OUT_0 = 1$, $IN_3 = 1$) devem permanecer neste estado durante um intervalo de tempo suficiente para essa propagação.

2.3. ESTRUTURA DE UM PROCESSADOR

Vias de Comunicação

As linhas que servem como meio de comunicação entre vários registros recebem o nome de vias ou barras (em inglês "BUS"). No exemplo anterior temos 1 via de oito bits a que chamamos de B.

Do exemplo deve ter ficado claro que uma via só pode conter uma informação a um determinado tempo, isto é, é impossível, ao mesmo tempo, transferir o conteúdo de R_0 para R_3 e de R_1 para R_2. Seria, no entanto, possível transferir informação de um registro para vários outros simultaneamente.

Estrutura de um Processador Simples

Suponhamos um processador que tenha quatro registros R_0, R_1, R_2 e R_3 para uso geral. Além de transferir dados de um registro para outro, o processador precisa também de ter capacidade para transformar esses dados através de operações aritméticas.

A Fig. 2.21 mostra uma possível arquitetura para este sistema.

O circuito denominado ALU é a unidade aritmética e lógica e tem a capacidade de efetuar uma operação aritmética (ex. soma, subtração) entre os dois operandos que se apresentam à sua entrada produzindo um resultado (resultado da operação aritmética) que, no exemplo, seria enviado para R_5 e indicadores (códigos que resumem o resultado da operação, "FLAGS") que, no exemplo, seriam armazenados em R_6.

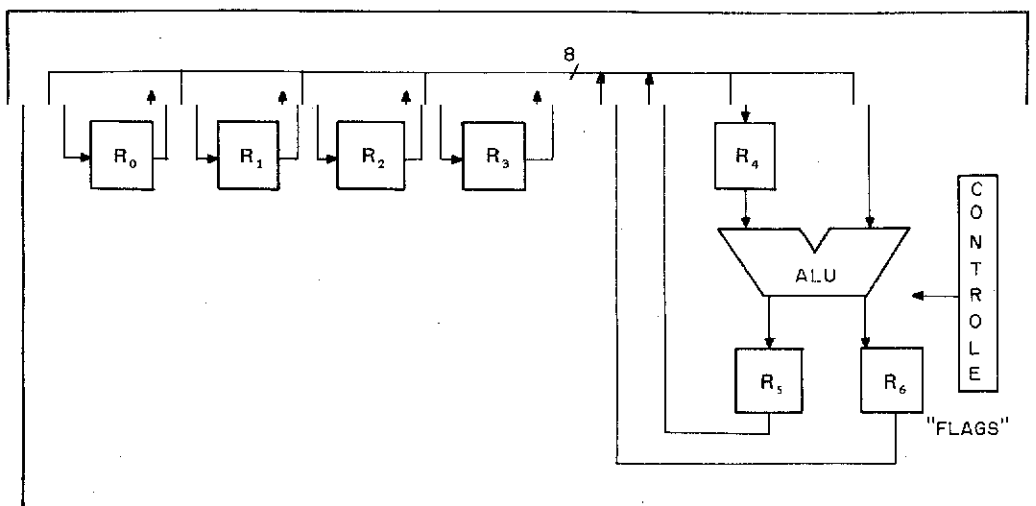



Fig. 2.21 — Processador simplificado.

Vemos que, além dos quatro registros iniciais, outros registros "auxiliares" foram utilizados. A necessidade dos registros auxiliares deve-se ao fato de termos somente uma VIA para transmitir os dados. Como a ALU precisa de dois operandos na sua entrada (e não é capaz de memorizá-los), precisamos enviar primeiro um operando para ser armazenado em R_4 e depois só o segundo operando que pode ser memorizado pelo próprio registro-fonte. O resultado da operação não pode se propagar direto à VIA porque isto causaria conflito com o registro-fonte e, por isso, deve ser armazenado em R_5 para, posteriormente, ser enviado ao registro de destino.

Exemplo:

Consideremos o problema de somar R_0 com R_1 e deixar o resultado em R_3. Sejam IN(N) e OUT(N) respectivamente as habilitações de entrada e de saída do registro R_n. Isto seria feito em três passos:

1º passo: (transferir R_0 para R_4)
OUT(0) = 1; IN(4) = 1
2º passo: (alimentar R_4 e R_1 na ALU e guardar resultado)
OUT(4) = 1; OUT(1) = 1; IN(5) = 1; IN(6) = 1
3º passo: OUT(5) = 1; IN(3) = 1

Em cada um dos passos acima, devemos observar o seguinte:

1. Todos os sinais que não foram explicitamente feitos iguais a 1 neste passo são iguais a zero. Assim sendo, no passo 2, OUT(0) é zero já que não foi feito explicitamente OUT(0) = 1 durante este passo.

2. Durante a execução de um passo, todos os sinais que serão ativados (feitos iguais a 1) são ativados simultaneamente no início e desativados simultaneamente no fim da execução deste passo.

3. A duração de um passo deve ser suficiente para que os dados se propaguem pelas vias e, se for o caso, pela unidade aritmética também. Assim sendo, supondo que o tempo de propagação pelas vias fosse de 50 ns e o tempo de propagação pela unidade aritmética fosse 40 ns, deveríamos ter as seguintes durações mínimas:

passo 1: 50 ns
passo 2: 50 + 40 = 90 ns
passo 3: 50 ns.

4. Como qualquer duração maior que a mínima pode ser utilizada, e existe uma conveniência de se ter todos os passos com a mesma duração, poderíamos:

a) usar 90 ns para duração de todos os passos;
b) desdobrar o passo 2 em dois passos idênticos durando 50 ns cada.

A alternativa (*a*) implica em um maior consumo de tempo (270 ns) ao passo que a alternativa (*b*) executa a mesma operação em 200 ns; (*b*) requer um passo a mais a ser executado o que, via de regra, é uma inconveniência desprezível.

Lógica de Controle

A operação vista acima, $R_3 \leftarrow R_0 + R_1$ é, tipicamente, uma *instrução* do computador.

Como foi visto no exemplo, a execução de uma instrução requer um certo número de passos aonde alguns *sinais de controle* são ativados. Estes sinais incluem habilitações de entrada e de saída e também outros tipos de controle como, por exemplo, para indicar à ALU se desejamos efetuar uma soma ou subtração.

Quem decide quais os sinais apropriados e qual o momento propício de ativar cada um deles?

A resposta é: um circuito denominado controle, lógica de controle, controlador ou seqüenciador. Existem dois modos comuns de se realizar um controlador: lógica aleatória e microprogramação.

Lógica aleatória. Este tipo de implementação é o mais antigo porém ainda é usado em poucos casos. Consiste em se fiar um circuito independente para cada instrução.

Microprogramação

Este tipo de implementação, que será melhor explorado na Seç. 2.6, "Exemplo: vamos fazer um processador", ao invés de requerer um circuito para cada operação, usa uma espécie de "programa" que indica, passo a passo, quais as entradas e quais as saídas de registros estarão habilitadas de modo a executar uma determinada operação do computador. A microprogramação é a implementação mais usada atualmente.

2.4. MEMÓRIAS

A memória é um circuito digital destinado a armazenar informações. Podemos idealizar uma memória pelo circuito da Fig. 2.22.

Cada um dos registros R_0 ... R_3 é capaz de armazenar um número (de 8 bits no caso) que lhes seja enviado através da linha de dados. Para que este dado seja armazenado, a habilitação de entrada do registro deverá estar aberta (em 1).

Pode-se também ler a informação contida em cada um dos registros colocando-se sua habilitação de saída em 1 pois desta forma os dados contidos no registro propagar-se-ão pela via de dados.

O controle das habilitações deve ser feito com base em duas informações: o endereço e a função desejada.

O endereço irá especificar qual o registro; isto será feito por um código de 2 bits que vem pela linha de endereços; o código passa por um decodificador que então ativa a linha correspondente ao registro desejado.

A função (ler ou escrever) será dada por duas outras linhas (MR e MW) e dirá se devemos abrir a habilitação de entrada (MW) ou saída (MR) do registro escolhido.

As portas "E" ("AND") à entrada das habilitações servem justamente para combinar a seleção do registro dada pelo decodificador com a função desejada.

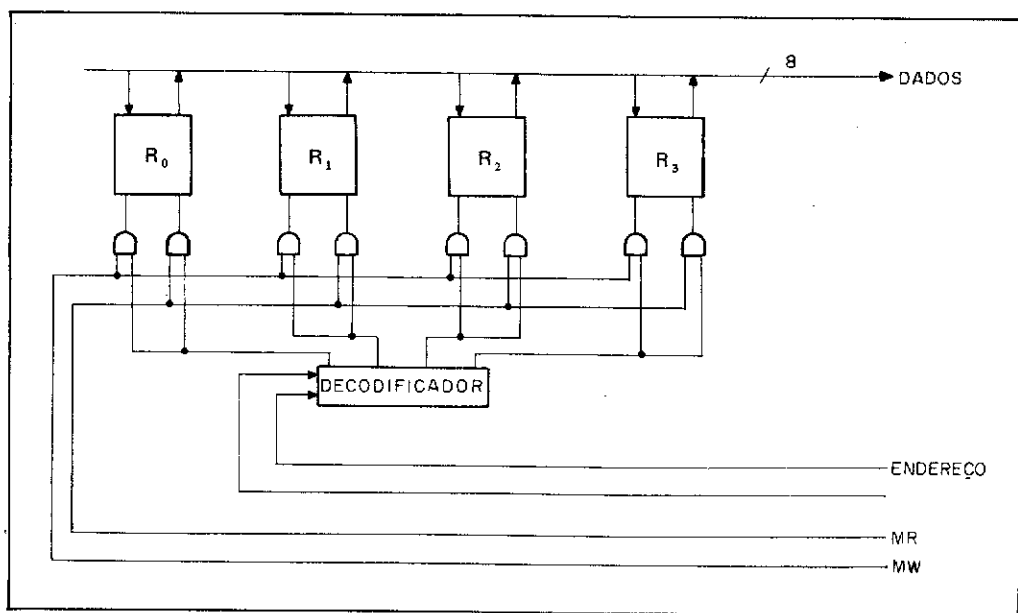



Fig. 2.22 – Memória.

As necessidades de memória de um computador são, normalmente, muito grandes porém este modelo idealizado abrange os conceitos básicos envolvidos.

Cada um dos registros é uma *posição de memória* especificada por um *endereço*. Devido ao tempo necessário para que os sinais se propaguem pelas vias e pela lógica de seleção (decodificação) a gravação ou leitura não é imediata; *Tempo de acesso* é o tempo mínimo que deve transcorrer entre um pedido de leitura e o momento em que o resultado é colocado na via de dados.

Durante uma leitura as linhas de endereço devem permanecer estáveis; durante uma gravação tanto as linhas de endereço como a de dados devem permanecer estáveis.

Como são as Memórias

Na Fig. 2.23 vemos quatro registros que podem ser usados como memória. Temos duas linhas de endereço E_1 e E_0 e duas linhas de controle: R (Read) e W (Write) além das linhas de dados D.

De acordo com o endereço presente nas linhas E_1 E_0 o decodificador deverá ativar apenas um dentre os quatro registros R_0, R_1, R_2 e R_3. No entanto, como a operação pode ser de leitura ou escrita, devemos cuidar que a entrada *ou* a saída do registro seja habilitada; ou seja, numa operação de escrita devemos habilitar a entrada (IN) do registro correspondente para que os dados porventura presentes nas linhas D possam "entrar" no registro; de modo análogo, numa operação de leitura apenas a saída (OUT) do registro correspondente deve ser habilitada, permitindo que os dados se propaguem para fora do registro e, posteriormente, pelas linhas de dados.

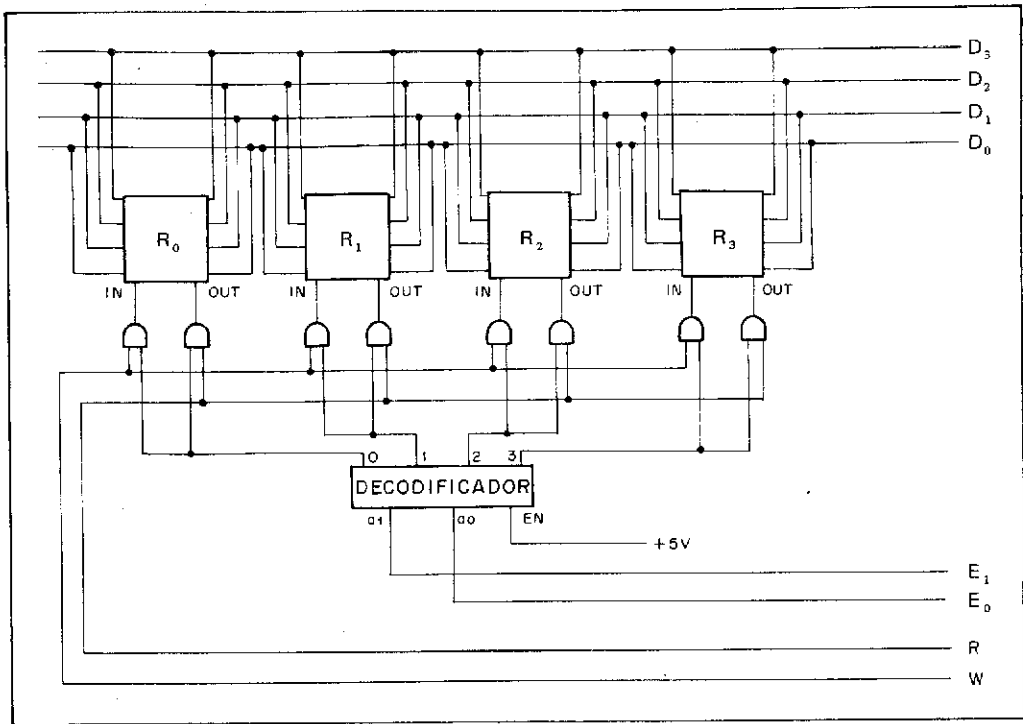



Fig. 2.23 – Exemplo de memória simples-1.

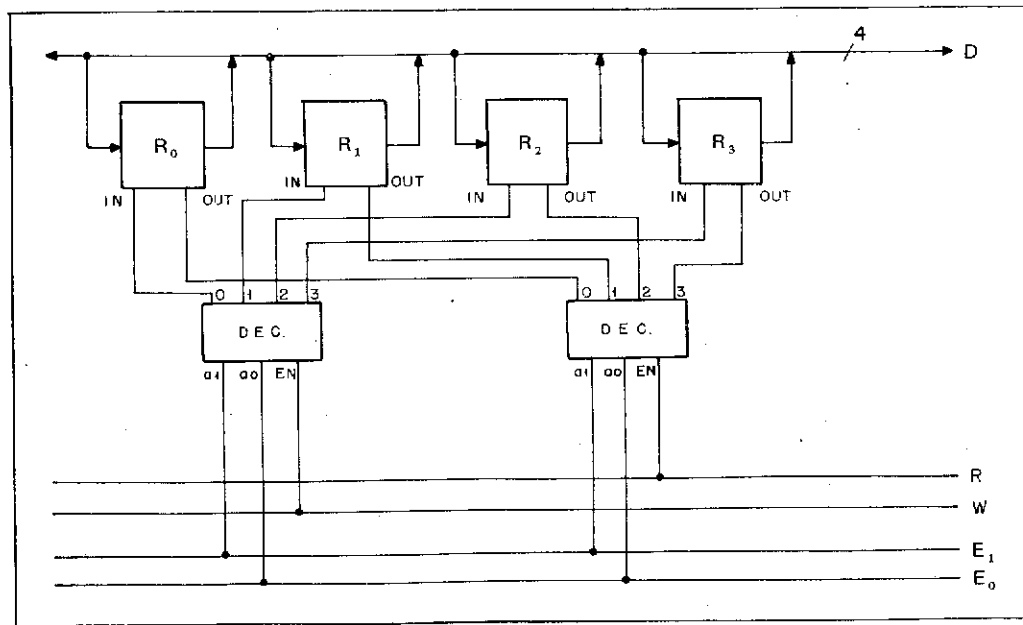



Fig. 2.24 – Exemplo de memória simples-2.

Para fazer esse controle podemos usar algumas portas "AND" como mostra a Fig. 2.23. Cada registro tem duas portas destas, uma ligada no "IN" e outra a "OUT". Todas as portas ligadas ao IN estão controladas pelo sinal W, todas as portas ligadas a OUT são controladas pelo sinal R. Assim, numa operação de escrita, (W = 1 R = 0) os OUT serão sempre desligados (já que R = 0); estará ligado apenas o IN correspondente ao registro endereçado.

Uma outra alternativa para obter o mesmo resultado é exposta na Fig. 2.24. Aqui são usados dois decodificadores ao invés de um com as portas AND. Neste caso, um decodificador está controlado pelo sinal R (Read) e outro pelo sinal W (Write). Como sabemos, um decodificador controlado terá todas as saídas em ZERO enquanto seu controle estiver desligado (em ZERO); assim o decodificador à dire'ıa somente funcionará quando R = 1 (leitura) e o da esquerda quando W = 1 (escrita).

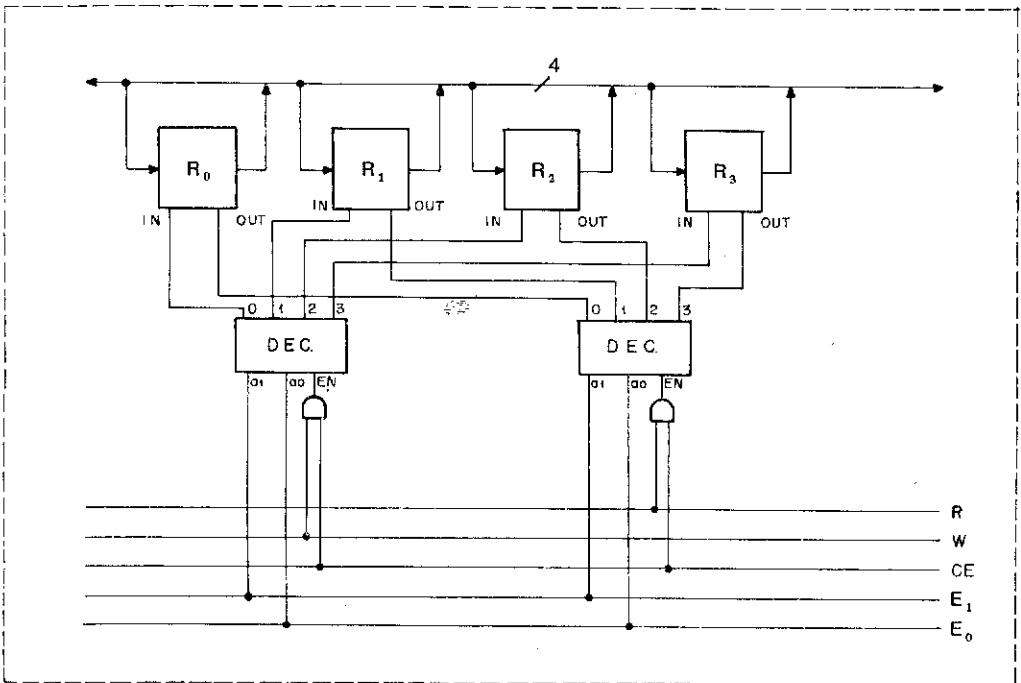



Fig. 2.25 — Uso da habitação CE.

CE — Chip Enable (Habilitação da Pastilha)

Existe um outro controle muito conveniente que nos permitiria desligar ambos os decodificadores, cujo propósito veremos adiante. Este sinal, a que daremos o nome de CE, pode ser ligado ao controle dos decodificadores conforme mostra a Fig. 2.25. Neste caso, quando CE = 0 nenhum dos dois decodificadores responde e o circuito estará totalmente desativado.

Encapsulamento

Podemos agora imaginar um circuito equivalente ao da Fig. 2.25 sendo miniaturizado e encapsulado numa pastilha de plástico. Apenas um contato para cada sinal seria visível por fora tal como tenta mostrar a Fig. 2.26.

Além dos sinais R, W e CE (de controle), dos sinais E (endereços) e D (dados) vemos ainda contatos para os sinais VCC e GND que são, respectivamente, a fonte de alimentação (geralmente +5V) e terra. Estes contatos são fisicamente necessários ao funcionamento dos circuitos porém, como não têm influência no projeto lógico, freqüentemente são omitidos.

Neste exemplo nos referimos a uma memória com 4 posições (4 registros) de 4 bits cada um, um total de 16 bits. Isto foi feito para facilidade didática pois a tecnologia atual pode condicionar entre 1 a 64K (K = 1024) bits em uma pastilha.

Obviamente uma pastilha com mais posições de memória deverá ter mais linhas de endereço para poder selecionar a posição desejada; no exemplo, para 4 posições usamos 2 linhas, para 8 posições usaríamos 3 linhas e assim por diante; para 2^N posições usaríamos N linhas.

Também, o número de linhas de dados irá variar correspondendo ao número de bits em cada posição de memória. Sendo muito comum 1 bit, 4 bits e 8 bits.

Na terminologia usual, uma memória 16K × 1 quer dizer 16K posições de 1 bit cada uma.

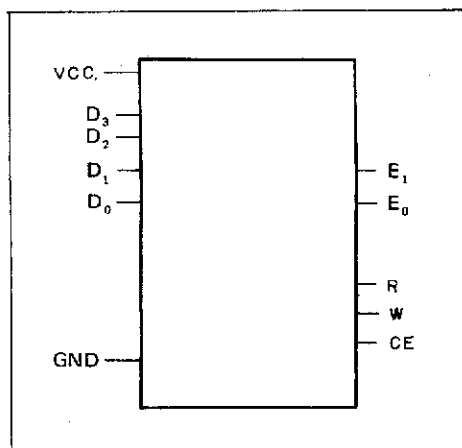



Fig. 2.26 — Representação de uma memória.

Expansão de Memórias

Normalmente são necessárias várias pastilhas para satisfazer os requisitos de memória de um sistema.

Novamente, a mesma estrutura básica encontrada no interior da pastilha será usada como mostra a Fig. 2.27. Uma via de dados, uma via de endereços, os sinais de controle e um decodificador.

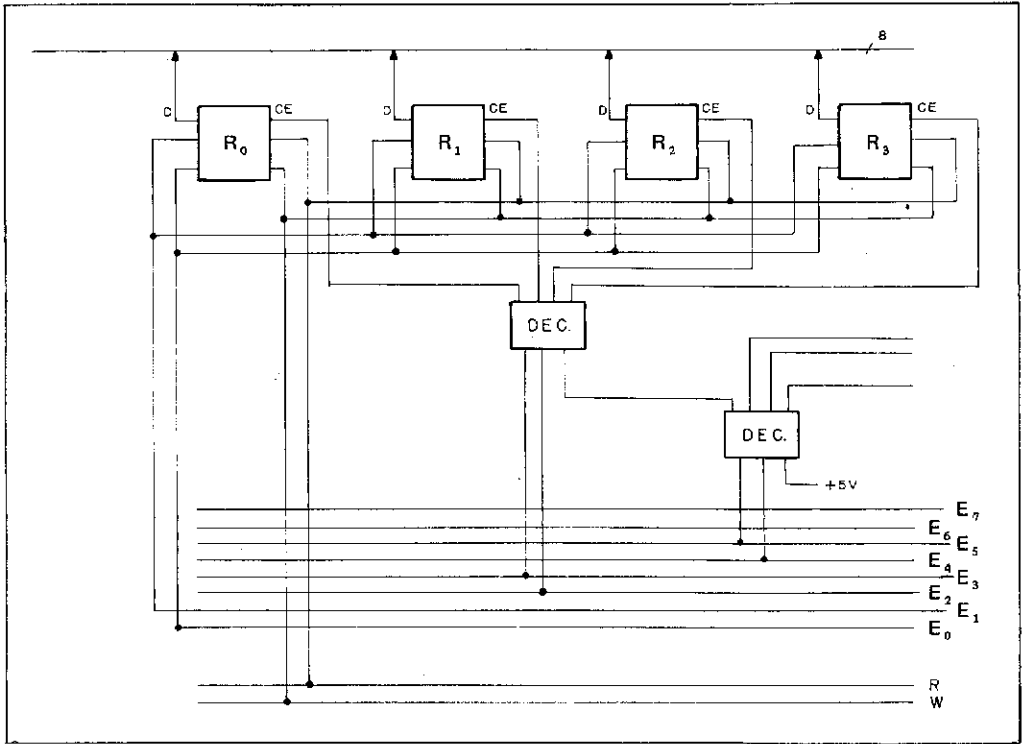



Fig. 2.27 – Exemplo de memória.

Usando essa estrutura, temos 4 vezes mais memória do que o disponível com uma só pastilha. Por que 4 vezes? Poderia ser mais? Isto depende apenas do decodificador disponível.

Memórias Semicondutoras

Atualmente as memórias fabricadas com material semicondutor são as mais utilizadas devido a seu baixo custo, pequeno tamanho, baixo consumo e vasto horizonte tecnológico.

Quanto à função, tais memórias podem ser classificadas em:

ROM — Memórias de leitura somente ("Read-Only Memories")
RAM — Memórias de leitura/gravação ("Random Access Memories")

ROM's

Na verdade a ROM também pode ser escrita (pois, de que adiantaria uma memória caso contrário?), porém essa escrita é um processo diferente do utilizado pelas RAM's e que, normalmente, implica em que o computador não consegue escrever na ROM. A gravação da ROM pode ser feita durante a fase final de fabricação das mesmas (Mask Programmable ROM) ou em alguns equipamentos especialmente construídos para esse fim usando ROM programável

(PROM). Em alguns casos é possível apagar e reescrever um novo conteúdo em uma ROM caso em que são denominadas EPROM sendo muito populares as EPROM's apagáveis mediante exposição à luz ultravioleta.

Obviamente, o custo da memória cresce com a flexibilidade; o tipo menos flexível é a ROM programada por máscara que só pode ser gravada na própria fábrica. O custo por unidade é baixo porém exigem-se grandes quantidades (mínimo 2 000) e não há possibilidade de se mudar a gravação. O tipo mais flexível encontrado no mercado é a EPROM que pode ser apagada e reescrita várias vezes usando-se um equipamento relativamente simples. Tipicamente o custo da EPROM é da ordem de 10 vezes o da ROM.

RAM's

A abreviatura RAM é inapropriada pois se refere a Memória de Acesso Aleatório, o que nada tem a ver com a possibilidade ou não de se gravar; porém a prática já consagrou RAM como sendo uma memória capaz de ser lida e escrita.

As RAM's semicondutoras apresentam uma característica desvantajosa que é a volatilidade; isto é, o armazenamento da informação depende de um suprimento contínuo de energia. Se a energia é interrompida, a informação da memória é destruída.

Uma classificação muito importante das RAM's é quanto ao modo de operação necessário para manter a informação. A RAM estática é fabricada com registros semelhantes aos do modelo idealizado a que nos referimos no início deste capítulo; os dados, uma vez armazenados, permanecem em um registro; assim, enquanto houver suprimento de energia, os dados são preservados.

As RAM's dinâmicas são implementadas com capacitores que são carregados com uma carga elétrica. Infelizmente, essa carga não permanece pois os capacitores vão aos poucos descarregando (como se houvesse um vazamento) e ao término de alguns milissegundos já não se pode mais determinar se o conteúdo original de determinado bit era 1 ou 0.

Deve-se observar, que, muito embora esses milissegundos possam parecer desprezíveis para nós, qualquer microprocessador seria capaz de executar alguns milhares de instruções antes que a informação desaparecesse. Em todo caso, é preciso fazer com que a informação possa ser retida por tempo indeterminado. Para isso basta que, periodicamente, a memória seja lida (e regravada).

Bem, mas para que tanta complicação? Não seria mais simples usar uma RAM estática?

Sim, o uso da RAM estática poderia ser mais simples apesar de dificilmente ser mais econômico. A grande vantagem da memória dinâmica advém justamente de usar elementos de circuitos (capacitores) que ocupam uma área muito menor que os elementos da memória estática (registros, flip-flops). Dessa forma, dentro da área normalmente disponível para se implementar um circuito integrado (inferior a 1 cm^2) é possível armazenar muito mais "bits" em capacitores do que em registros.

Como ilustração podemos citar que as RAM's estáticas mais modernas tipicamente podem conter 8K bits (Intel 8185) ao passo que as RAM's dinâmicas contêm 64K bits, isto é, oito vezes mais informação.

Em matéria de custo também as RAM's dinâmicas levam vantagem e este é o argumento definitivo a seu favor. O custo por bit da RAM dinâmica é tão mais baixo que compensa o custo adicional do circuito extra para "refrescar" a

memória. Para melhorar ainda mais essa vantagem, já começaram a aparecer memórias dinâmicas que já contêm o circuito de "refresco" na própria pastilha.

2.5. ENTRADA/SAÍDA (Input-Output Ports)

Como sabemos, o computador tem necessidade não somente de armazenar e consultar informações na memória como também de trocá-las com o mundo exterior. Esta troca é normalmente feita usando-se circuitos denominados portas de entrada (só recebem informações do exterior), portas de saída (só enviam informações para o exterior) ou portas de entrada/saída (podem funcionar nos dois sentidos). Tais portas nada mais são do que os já familiares "depósitos" como ilustra a Fig. 2.28.

Vemos aí oito depósitos sendo que P_0 P_1 P_2 P_3 têm suas entradas ligadas à via de dados (podendo receber dados do computador) e P_0' P_1' P_2' P_3' têm suas saídas ligadas à via de dados (podendo enviar dados ao computador).

P_0 a P_3 funcionarão como portas de saída (porque recebem dados que saem do computador); suas linhas de saída podem ser usadas para comandar eventos exteriores (acender luzes, ligar motores etc.). P_0' a P_3' são portas de entrada (porque recebem dados que entrarão no computador).

A exemplo do que foi ilustrado com as memórias, usamos dois decodificadores, um acionado pela linha R (leitura) que comanda as portas de entrada; outro acionado pela linha W (escrita) que comanda as portas de saída.

Saída de Dados

Assim, quando o computador quer uma operação de saída, para a porta 2 por exemplo, ele envia o endereço da porta desejada (10), coloca os dados na via de dados e, finalmente, liga o sinal W. Este sinal ativa o decodificador que, por seu turno, habilita a entrada do depósito (P_2). A informação presente na linha de dados é armazenada no depósito e se propaga pela saída, já que esta permanece sempre ligada. Assim, esta porta manterá em sua saída a mesma configuração de bits até que o computador escreva nova informação nesta mesma porta.

Entrada de Dados

A entrada de dados far-se-á de modo semelhante. Note-se, porém, um pequeno "truque" na habilitação de entrada dessas portas (P_0' a P_3'). A habilitação poderia ficar permanentemente ligada (simetricamente ao que é feito com as portas de saída). Dessa forma, mudanças nos sinais provenientes do exterior seriam registradas imediatamente. Existe, porém, o inconveniente de que os dados mudem durante a leitura.

Isto pode ser evitado controlando-se a entrada com o sinal de leitura (R) invertido. Assim, os dados exteriores podem variar sendo registrados enquanto

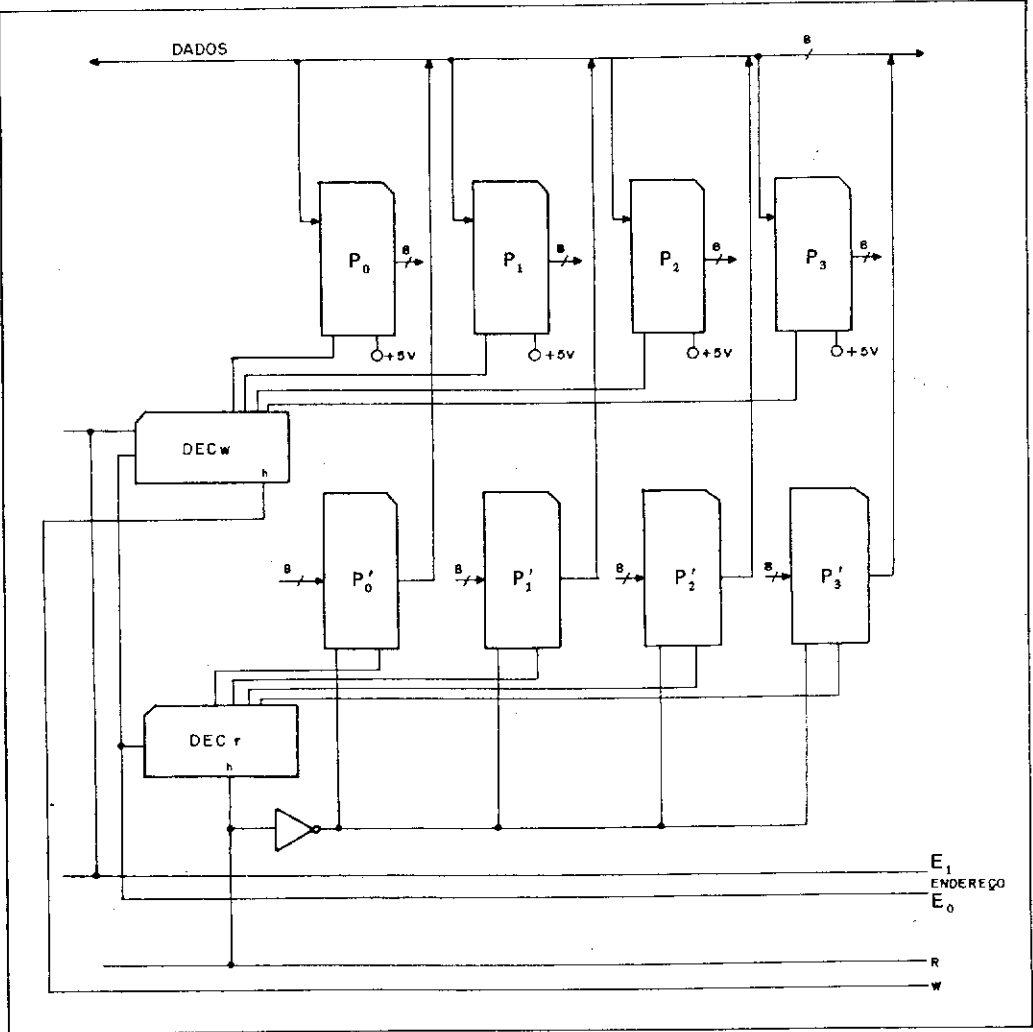



Fig. 2.28 – Portas de Entrada e Saída.

não houver leitura. Durante a leitura (R = 1) a configuração presente na entrada seria "fotografada" pela porta e esta "fotografia" seria mostrada ao processador (dando uma "imagem nítida" e não "tremida"). Caso haja mudança na entrada, somente será registrada no fim da leitura.

2.6. EXEMPLO: VAMOS FAZER UM PROCESSADOR?

A Fig. 2.29 mostra o *hardware* básico para se implementar um processador simples. Há um conjunto de registros-depósito numerados de R_0 a R_9 entre os

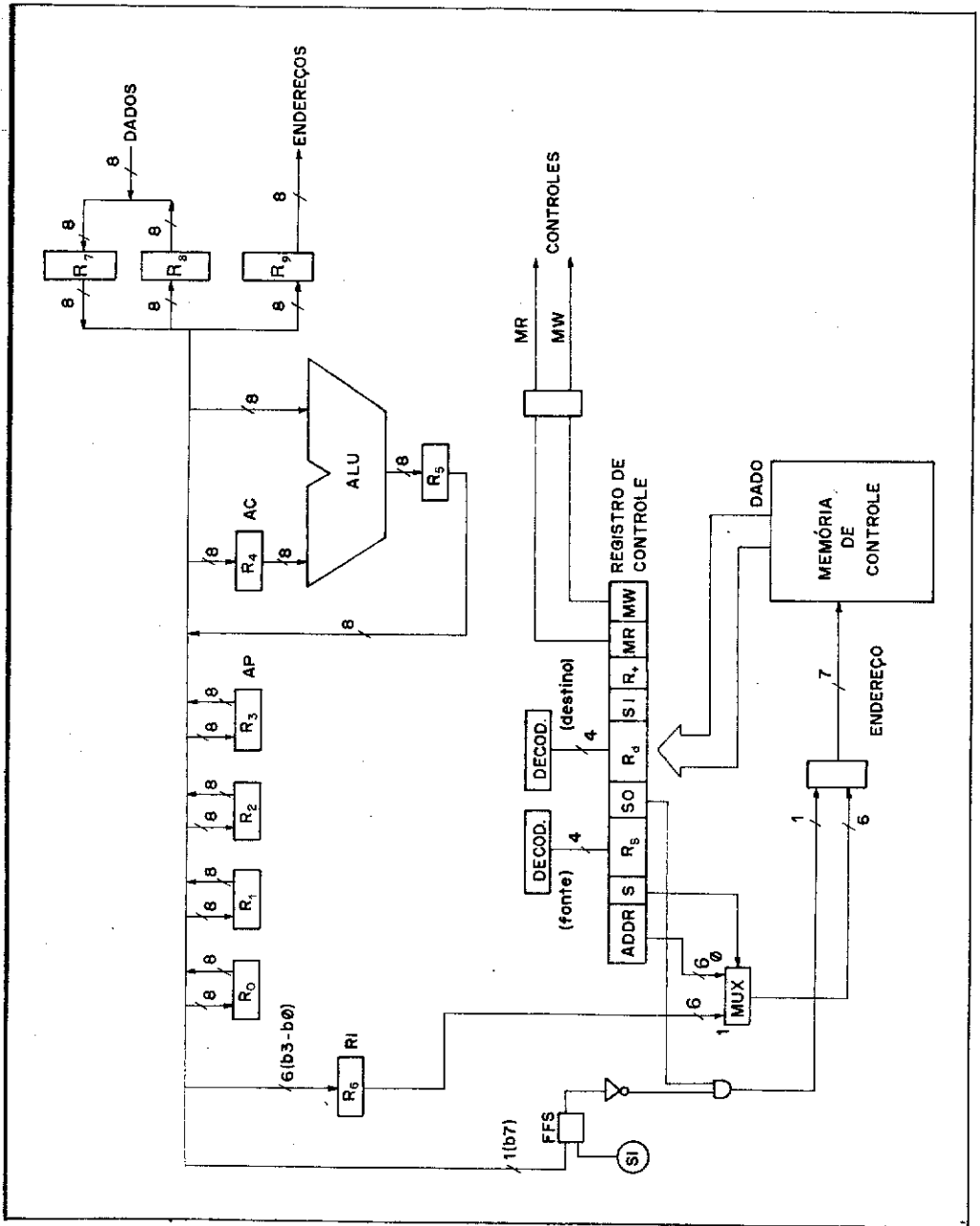



Fig. 2.29 — Esquema de um processador.

quais é possível transferir dados controlando-se suas habilitações de entrada e de saída. As conexões destas habilitações foram omitidas para simplificar a figura, porém as saídas do decodificador fonte (à esquerda) estão ligadas, respectivamente, as habilitações de saída dos registros R_0 a R_9 e as saídas do decodificador destino (à direita) estão ligadas às habilitações de entrada dos mesmos registros.

A movimentação de dados entre registros é então feita do registro-fonte (campo R_s do registro de controle) para o registro-destino (R_d do registro de controle), assim para transferir dados de R_1 para R_4, devemos cuidar para que R_s contenha o valor 1 e R_d o valor 4.

A unidade aritmética — ALU. Em geral teríamos aqui uma unidade capaz de executar vários tipos de operação porém, para o nosso exemplo, basta considerarmos um simples somador.

Os registros. Os registros R_0, R_1 e R_2 são de propósito geral; R_3 será utilizado como apontador do programa (Ap), por isso, como veremos adiante, será possível incrementá-lo sem usar a unidade aritmética. R_4 poderá ser usado como um acumulador (Ac) devido à sua localização logo à entrada da unidade aritmética. R_5 é um registro auxiliar necessário para guardar o resultado da operação aritmética.

R_6 é o registro de instrução (RI) no qual deverá ser guardado o código de operação da instrução que está sendo executada.

Os registros R_7, R_8 e R_9 destinam-se a comunicação com a memória e/ou portas de entrada/saída. R_9 tem sua entrada ligada à via interna do processador e sua saída ligada à via de endereços do sistema. Um valor colocado em R_9 será propagado pela via de endereços e, dessa forma, será visto pela memória e demais componentes do sistema. Os registros R_7 e R_8 são intermediários entre o processador e a via de dados: R_7 é usado para guardar dados que cheguem por esta via e R_8 é usado para mandar dados pela via. A operação de leitura com a memória é portanto semelhante ao que já foi visto anteriormente: o processador coloca o endereço desejado na via de endereços (ou seja, coloca o endereço em R_9) e liga o sinal de leitura MR (MR = 1). Transcorrido o tempo necessário, o resultado deverá estar presente na via de dados bastando então aprisioná-lo em R_7 de onde poderá ser transferido para qualquer outro registro.

O registro de controle será utilizado para conter a microinstrução em andamento, isto é, todos os controles, inclusive habilitações de registros que estarão ligados durante um tempo determinado. Este registro é carregado a partir da memória de controle e consiste de vários campos descritos a seguir:

Addr — (6 bits) — é o endereço da próxima microinstrução a ser executada (as microinstruções não são executadas seqüencialmente).

S — (1 bit) — indica se o endereço da próxima microinstrução deve proceder do campo Addr (S = 0) ou do RI (S = 1).

R_s — (4 bits) — indica registro-fonte, isto é, o número de um registro cuja saída será habilitada para a via interna.

SO — (1 bit) — habilita a saída do "flip-flop" de sinal.

R_d — (4 bits) — indica registro-destino, isto é, o número de um registro cuja entrada será habilitada.

SI — (1 bit) — habilita a entrada do "flip-flop" de sinal, isto é, se SI = 1, o bit mais significativo (bit de sinal) presente na via de dados será armazenado no registro FFS.

R_+ — (1 bit) Se $R_+ = 1$, R_3 será incrementado.

MR — (1 bit) ler da memória — este bit estará diretamente ligado à via de controle, indicando que se deseja uma leitura da posição de memória cujo endereço deve estar na via de endereços.

MW — (1 bit) escrever na memória — este bit estará ligado à via de controle, indicando que se deseja uma escrita na memória, ou seja, o conteúdo das linhas de dados deve ser escrito na posição de memória cujo endereço é especificado nas linhas de endereço.

1. O conteúdo da via de endereços deve estar estável enquanto MR = 1.
2. O conteúdo de endereços e dados deve estar estável enquanto MW = 1.

Observações

3. Os registros R_4 e R_9 têm as suas saídas sempre habilitadas.
4. O endereço da memória de controle é de 7 bits, sendo o bit mais significativo igual ao sinal do último resultado, no caso da instrução de desvio condicional e "0" nos outros casos; os outros 6 bits ou vêm do registro de instrução ou do campo Addr do registro de microinstrução.

Mecanismo de Controle. O funcionamento do sistema baseia-se num circuito (não incluído na figura) que, ao final de cada intervalo de tempo T, aciona a memória de controle e carrega o resultado desta leitura no registro de controle que permanecerá com este valor até a próxima operação.

Ao inicializar a operação do sistema, usa-se um circuito capaz de zerar todos os registros; dessa forma, o endereço que aparece na entrada da memória de controle será zero. Fazendo com que a primeira microinstrução a ser executada seja a que está contida no endereço zero. Esta microinstrução deverá ser o início da operação de busca (fetch), responsável por apanhar a próxima instrução a ser executada na memória do computador (como inicialmente o Apontador de Programa também é igual a zero, conclui-se que automaticamente executaremos a instrução que está na posição zero da memória principal).

Vamos assumir que as instruções ocupem sempre duas posições: a primeira contendo o código da operação (carregar, somar, desviar etc.) e a segunda o endereço do operando (valor a ser carregado ou somado, endereço do desvio etc.).

Esta rotina de busca obviamente necessitará de mais de uma microinstrução e, ao seu final, deverá ter colocado em R_6 (RI) o código da operação. Note-se que R_6 está ligado ao endereço da memória de controle, sendo um procedimento muito simples desviar de acordo com o valor de R_6.

A memória de controle. A memória de controle será usada para conter o microprograma do sistema, ou melhor, os microprogramas (se denominarmos por microprograma o conjunto de microinstruções necessários a execução de cada instrução do computador).

Ao contrário dos programas a que estamos habituados os microprogramas não são, em geral, executados seqüencialmente. Normalmente, cada microinstrução especifica, de alguma forma, qual a próxima microinstrução a ser executada. Assim, é possível organizar a memória de controle de modo a que haja uma correspondência muito simples entre o código de operação e o endereço de início do correspondente microprograma (queremos, com isso, dizer que uma instrução cujo código de operação seja 03 terá seu microprograma iniciando na posição 03 da memória de controle).

O Microprograma de Busca. O microprograma de busca deve:

- ler o código de operação da próxima instrução da memória e trazê-lo para R_6;

- atualizar o apontador de programa;
- ler o campo de endereço dessa instrução (podendo deixá-lo em R_7);
- atualizar o apontador de programa.

Vamos assumir que T = 100 ns (isto quer dizer que 100 ns deve ser suficiente para propagar todos os sinais dentro do processador) e que a memória responda em 250 ns.

O microprograma da busca poderia ocupar as posições 0 e de 16 a 23 da memória de controle conforme a Fig. 2.30. As posições de 1 a 15 poderiam ser

	ADDR	S	R_s	SO	R_d	SI	R_+	MR	MW	Observações
0	16	1	3	0	9	0	0	0	0	$R_9 \leftarrow R_3$
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										
11										
12										
13										
14										
15										
16	17	1	15	0	15	0	1	1	0	$R_3 \leftarrow R_3 + 1$; MR = 1
17	18	1	15	0	15	0	0	1	0	MR = 1;
18	19	1	15	0	7	0	0	1	0	$R_7 \leftarrow$ dados; MR = 1
19	20	1	6	0	7	0	0	0	0	$R_6 \leftarrow R_7$;
20	21	1	3	0	9	0	0	0	0	$R_9 \leftarrow R_3$;
21	22	1	15	0	15	0	1	1	0	$R_3 \leftarrow R_3 + 1$; MR = 1
22	23	1	15	0	15	0	0	1	0	MR = 1;
23	00	0	15	0	7	0	0	1	0	$R_7 \leftarrow$ dados; MR = 1
24										
25										
26										
27										
28										
29										
30										

Fig. 2.30 — Mapa da memória de controle.

usadas para iniciar o microprograma de instruções do computador (carregar, somar etc.). O leitor é altamente encorajado a tentar elaborar pelo menos o microprograma da instrução de soma.

Note-se que todas as microinstruções exceto a última (posição 23) contêm o campo S = 1. Isto porque cada uma especifica no campo ADDR o endereço da microinstrução seguinte; porém, terminada a busca o endereço da próxima microinstrução será determinado pelo registro R_6 sendo o campo ADDR irrelevante neste caso (está igual a zero porque não pode ser deixado em branco). Assim, se $R_6 = 1$ será executada a microinstrução do endereço 1, se $R_6 = 2$ a do endereço 2 e assim por diante.

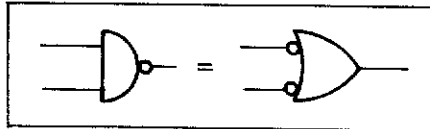
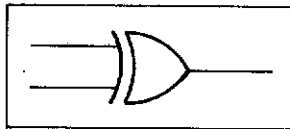
Conclusões. Este exemplo pode ser trabalhado pelo leitor interessado para incluir diversas instruções que não custam absolutamente nada além de espaço na memória de controle. Com ele procuramos reforçar que os computadores, tanto na memória, como na entrada/saída e como no processador consistem apenas de registros e ligações convenientes.

As memórias de controle são preenchidas normalmente pelo fabricante havendo alguns casos raros em que é possível sua alteração pelo usuário.

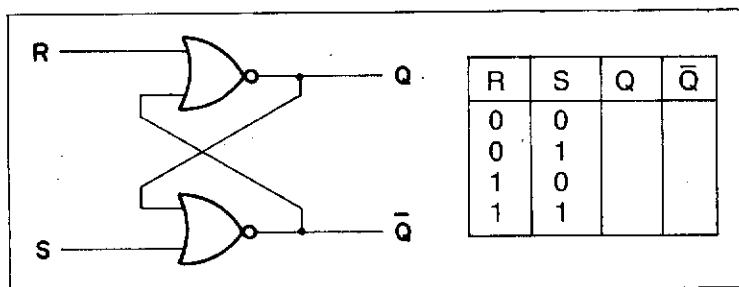
Acreditamos que o leitor não terá dificuldades de expandir o raciocínio apresentado para entender implementações mais complexas.

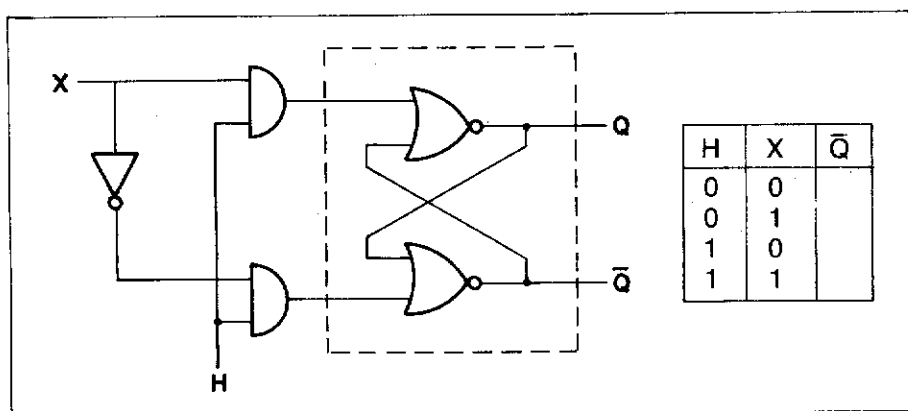
2.7. EXERCÍCIOS

1. Verifique que as duas representações de um "E invertido" (NAND) mostradas abaixo são equivalentes.

Sugestão: Como só existem 4 possíveis configurações de entrada pode-se verificar a saída nas duas representações em cada caso.

2. O bloco "OU exclusivo" tem a sua saída igual a 1 se uma das entradas (mas não ambas) for 1. Construa um "OU exclusivo" usando blocos "E", "OU" e "inversores".

3. Construa um "OU exclusivo" usando apenas blocos "E-invertido".

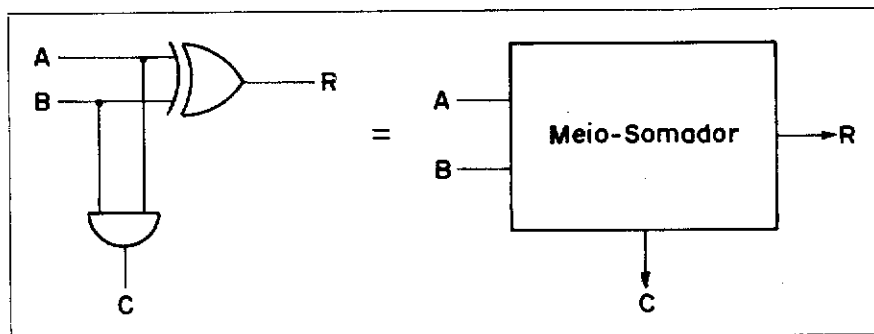
4. O circuito abaixo é o que se denomina um "flip-flop" do tipo R-S. As entradas são R e S e as saídas são Q e $\overline{Q}$. Complete a tabela com os valores de Q e $\overline{Q}$ para cada configuração de entrada R-S. Observe que quando R = 0 e S = 0 o valor das saídas Q e $\overline{Q}$ não será alterado.

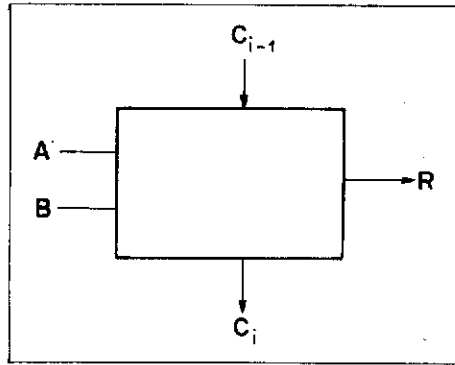
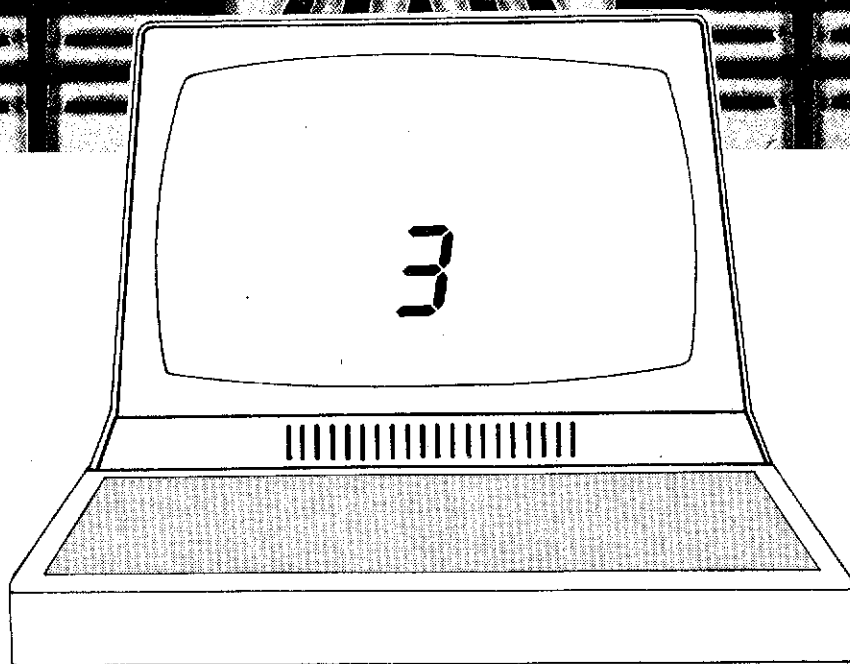
R	S	Q	$\overline{Q}$
0	0		
0	1		
1	0		
1	1		

5. O circuito abaixo é um "flip-flop" com habilitação de entrada; a parte contida no quadro pontilhado nada mais é do que um "flip-flop" R.S (exerçício 4).

H	X	$\overline{Q}$
0	0	
0	1	
1	0	
1	1	

a) complete a tabela dando o valor de $\overline{Q}$ para cada configuração de X e H;

b) explique como se pode utilizar esse circuito como uma memória para guardar o valor de um sinal X.

6. A partir do "flip-flop" do exercício 5, construa um registro de depósito de dados de 8 bits.

7. Um "meio-somador", ilustrado abaixo, é um circuito que executa a soma binária de dois sinais de entrada A e B dando um resultado R e um indicador de "vai um" C.

a) verifique que o resultado da operação é coerente com a tabuada binária;

b) mostre como usar dois meios somadores para construir um somador completo (o somador completo além das entradas A e B tem uma entrada para "vai um" do resultado anterior).

8. Mostre como usar somadores completos para construir um somador de 8 bits.

DESCRIÇÃO DOS MICROPROCESSADORES

3.1. MICROPROCESSADOR INTEL 8080/8085

O sistema 8085 foi introduzido no mercado pela INTEL no fim da década de 70 como sucessor do popularíssimo 8080 e tem rapidamente ganho terreno a ponto de que, desde 1979, raríssimos novos projetos utilizam o 8080.

No Brasil o sistema 8080 ainda é bastante popular inclusive em novos projetos. No entanto, essa popularidade se deve mais à relativa facilidade com que os componentes da família 8080 são encontrados no mercado do que ao seu próprio mérito e, possivelmente, do que ao desconhecimento de nossos projetistas. É de se esperar que esta situação evolua e que o 8085, seja, também no Brasil, o microprocessador de 8 bits mais popular.

O sistema 8085 tem, exatamente, o mesmo conjunto de instruções do 8080 acrescido de duas instruções (RIM e SIM). Isto permite o total aproveitamento de qualquer *software* desenvolvido e testado no 8080 sem qualquer modificação.

Devido a ser de tecnologia mais recente o 8085 também é mais rápido e incorpora mais funções na própria pastilha do processador. (Há pelo menos duas pastilhas que não são necessárias no 8085: o gerador de clock e o controlador das vias.) Outra vantagem muito importante é que o 8085 requer apenas uma tensão de alimentação (+5V) em contrapartida às 3 do 8080 (+12V, +5V, −5V).

O 8085 também é capaz de lidar com 3 interrupções diferentes sem necessitar de nenhuma pastilha externa. Finalmente, para dar uma vantagem definitiva ao 8085, foram projetadas e lançadas no mercado outras duas pastilhas incorporando múltiplas funções e que permitem a um sistema que conste de apenas 3 pastilhas atender a uma vasta gama de aplicações, pois este sistema dispõe de:

- — processador
- — 2K bytes de ROM ou EPROM
- — 256 bytes de RAM
- — 1 relógio ("Timer")
- — 38 linhas de entrada/saída
- — 3 entradas de interrupção

Arquitetura para o Programador

A arquitetura visível ao programador do 8085 é apresentada na Fig. 3.1. Existe um registro acumulador de 8 bits (A) onde são, normalmente, efetuadas as operações aritméticas e lógicas. Um registro (F) que contém códigos indicadores que resultam de algumas operações aritméticas e lógicas ("Flags"), outros 6 registros de 8 bits (B, C, D, E, H, L) e dois registros especiais de 16 bits: SP — o apontador da pilha ("Stack pointer") e PC — o apontador do programa ("Program Counter").

Pode-se endereçar até 64K bytes (K = 1024) de memória e até 256 portas de entrada/saída que farão a interface com o mundo exterior.

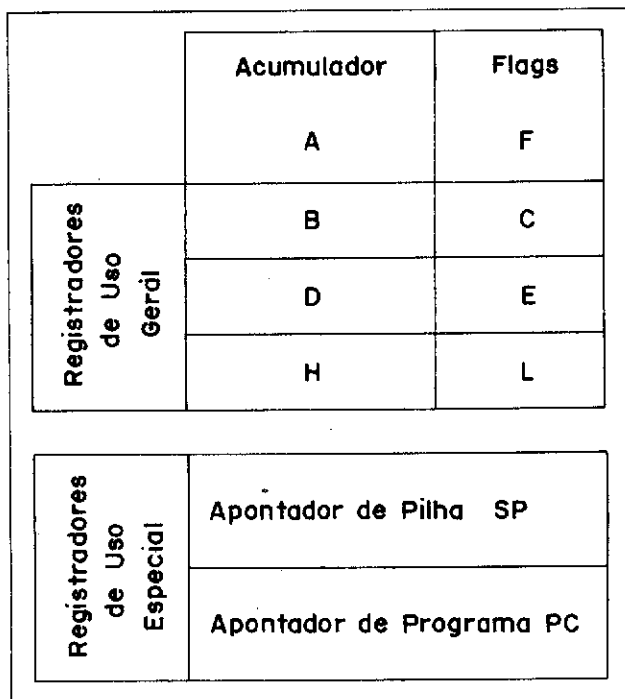



Fig. 3.1 — Registradores 8080A/8085.

As operações aritméticas (soma e subtração), lógicas (E, OU, OU-exclusivo, Comparação) e de entrada/saída se utilizam, normalmente, de dados que estão no acumulador e deixam o resultado da operação também no acumulador.

Os seis registros de 8 bits (B, C, D, E, H, L) servem para armazenar operandos temporariamente. É possível incrementar (somar 1) ou decrementar (subtrair 1) cada um deles individualmente. Usando algumas instruções especiais, é possível considerar um par (B, C), (D, E), (H, L) como um único registro de 16 bits.

O par de registros H, L tem duas funções especiais:

Pode funcionar como acumulador para a instrução de soma dupla (DAD); isto é, pode-se somar um operando de 16 bits armazenado em um par de registros (B, C), (D, E), (H, L) com o operando contido em (H, L) e deixando o resultado da operação em (H, L).

Pode funcionar como um apontador da memória contendo um endereço de 16 bits. Algumas instruções farão referência à memória referindo-se sempre ao endereço de memória contido em (H, L).

Por exemplo, ADD M (somar com a memória) será interpretado como somar o Acumulador com um número que está na memória, cujo endereço está nos registros H e L. O resultado fica no acumulador.

Sempre que nos referimos a um par de registros, entende-se que o 1º registro do par é colocado à esquerda (e contém os bits mais significativos). Os pares existentes são apenas (B, C), (D, E) e (H, L); não se pode formar outros tais como (C, E), (B, H) etc.

SP — Apontador da Pilha. O registro apontador da pilha ("Stack Pointer") é um registro de 16 bits que contém um endereço de memória que corresponde ao topo da pilha. A pilha é uma área de memória usada para armazenar dados de forma peculiar: normalmente queremos ler os dados na ordem inversa em que foram armazenados; isto é, se armazenamos três números na ordem cronológica A, B, C, retirá-los-emos na ordem C, B, A.

O nome "pilha" advém da semelhança com o fato de que os dados estariam sendo "empilhados" uns sobre os outros, não sendo possível retirar um dado a não ser do topo da pilha.

As pilhas são estruturas de dados muito úteis em computação e o leitor iniciante deve se referir a outras bibliografias especializadas.

No 8085, cada vez que um operando é posto na pilha, o SP é, automaticamente, decrementado de 1 para conter o novo endereço do topo da pilha (a pilha cresce do endereço mais alto para o mais baixo) e o operando é armazenado na posição de memória cujo endereço está no SP. Analogamente, quando se retira um operando da pilha, ele é retirado da posição de memória cujo endereço está no SP e o SP é, automaticamente, incrementado.

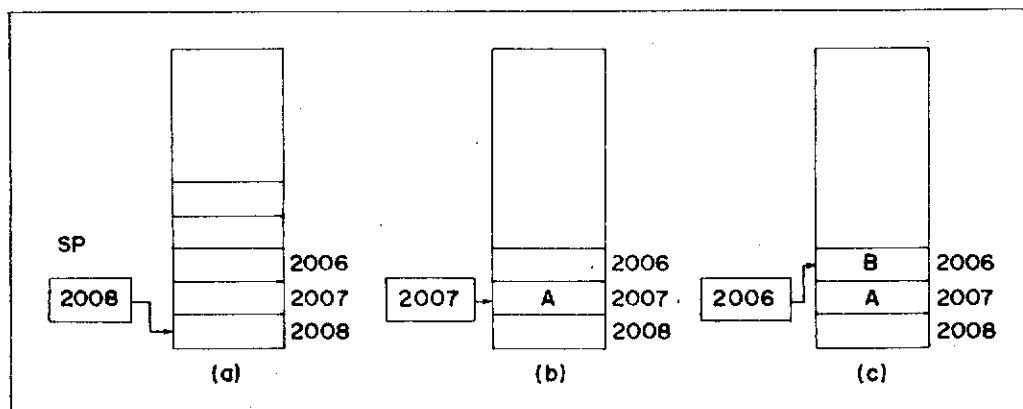



Fig. 3.2 — Pilha.

Na Fig. 3.2, seja SP = 2008. Se empilharmos um operando A, este será armazenado em 2007 que será o novo SP. Se empilharmos outro operando B, este será armazenado em 2006 se então, desempilharmos, recebemos o operando B que estava no topo da pilha; e o SP passará a ser 2007; se novamente desempilharmos receberemos o operando A e SP voltará a ser 2008.

PC — Apontador do Programa. O registro PC, apontador do programa, também é um registro de 16 bits com uma função muito especial: ele armazena um endereço da memória onde está a próxima instrução a ser executada.

Ao se iniciar o ciclo de cada instrução o processador sempre envia o conteúdo do PC para a memória pedindo-lhe a próxima instrução que será então decodificada e executada. (Este procedimento é chamado de "fetch".) Sempre que ocorre um desvio no programa, simplesmente este novo endereço é colocado no PC e a execução prossegue normalmente.

FLAGS. Os indicadores "Flags" são afetados pelas operações aritméticas e lógicas. Cada bit indicador tem um significado diferente, a saber:

Z — (zero)	—	Indica que o resultado é zero (ou, no caso de uma comparação, que os operandos são iguais).
S — (sinal)	—	Este bit coincide com o bit mais significativo do resultado e que, na notação complemento a 2, é zero para um número positivo e 1 para um número negativo.
P — (paridade)	—	Se o acumulador contiver um número par de bits em 1, este bit de paridade será 1. Caso contrário, será zero; ou seja, o número total de bits em 1 no conjunto acumulador + paridade deve ser sempre ímpar.
CY — ("Carry")	—	"Vai um" — este bit indica se houve propagação (vai um) a partir do bit mais significativo do acumulador.
A_c — ("Auxiliary Carry")	—	"Vai um auxiliar" — este bit indica se houve propagação do bit 3 para o bit 4 do acumulador.

É importante observar que:

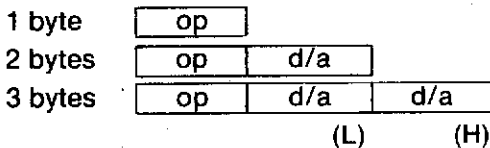
Numeramos os bits dentro de um registro da direita para a esquerda; isto é, o bit 0 é o menos significativo e o bit 7 (de sinal) é o mais significativo.

Nem todas as instruções afetam os indicadores.

O programador deve ler a descrição de cada instrução para confirmar o efeito da mesma.

Tipos de Instruções

As instruções 8080/8085 possuem três formatos distintos, podendo ocupar 1, 2 ou 3 bytes. O código de operação está sempre no 1º byte (pois assim o processador decide se já pode executar a instrução ou se deve buscar os demais bytes).

1 byte	op		
2 bytes	op	d/a	
3 bytes	op	d/a	d/a
		(L)	(H)

Nas instruções de 1 byte, o operando, se existir, faz parte do próprio código de operação e estará, necessariamente, especificado através de um registro.

Nas instruções de 2 bytes, o 2º byte pode conter um operando imediato (isto é, o próprio dado a ser operado) ou um endereço de 8 bits referente a uma porta de entrada/saída.

Nas instruções de 3 bytes o 2º e o 3º bytes são encarados como uma quantidade de 16 bits podendo ser também um operando imediato ou um endereço de uma posição de memória.

Considerando-se uma quantidade de 16 bits armazenada na memória seria natural ter o 1º byte (endereço mais baixo) contendo os bits mais significativos pois assim o conjunto byte 1, byte 2 seria considerado nesta ordem. Em nome da compatibilidade com sistemas anteriores (8008), os sistemas 8008, 8080, 8085 e Z80 consideram sempre que o 1º byte contém os bits menos significativos e, assim, o número 2010 seria armazenado na memória como 1020. Tal observação vale também para as instruções de 3 bytes. Por exemplo, a instrução JMP 2010 (desvie para 2010) se estivesse armazenada na posição 3000 de memória estaria assim:

Tipos de Endereçamento

Existem várias maneiras para se endereçar os dados em um microcomputador. Endereçamento é o modo pelo qual é possível as instruções referenciarem os seus operandos. O modo mais comum de endereçamento é o denominado DIRETO, que consiste em especificar na própria instrução o endereço da posição de memória onde o operando se encontra. No 8080/8085 temos os seguintes tipos de endereçamento:

— Registro
— Imediato
— Direto
— Registro indireto

Endereçamento de Registro. O endereçamento de registro é utilizado quando o operando está em um dos registradores do microprocessador, neste caso o registro será especificado no próprio código de operação, isto é, está contido no primeiro byte de instrução.

Exemplo: MOV A,B (mover do registro B para o registro A) os dois operandos estão em registros.

MOV A,B

Endereçamento Imediato. O endereçamento imediato é caracterizado pelo fato de a própria instrução já conter o operando, não sendo necessário especificar nenhum endereço. As instruções de endereçamento imediato são de 2 ou 3 bytes, sendo o segundo ou o segundo e o terceiro o operando, em função do operando ser de 1 ou 2 bytes.

Exemplo: MVI A,5 (mover o valor 5 para A) o 2º operando é imediato.

MVI A
05

Endereçamento Direto. O endereçamento direto exige instruções de 3 bytes, e o segundo e o terceiro byte da instrução são utilizados para conter o endereço de memória onde se encontra o operando.

Exemplo: LDA 0503 (carregar o acumulador com o conteúdo da posição de memória 0503).

LDA
03
05

Endereçamento Registro Indireto. No endereçamento registro indireto, a instrução está se referindo a um registro (no caso um par de registros), porém esse registro não contém o operando, mas sim um endereço de memória onde se encontra o operando.

Conjunto de Instruções 8080/8085

A seguir apresentamos uma descrição completa das instruções 8080/8085, sendo utilizada a seguinte nomenclatura:

Símbolo	*Descrição*
ACC, acumulador	Registro A
end	Endereço, 16 bits
e	dado, 8 bits
e16	dado, 16 bits
r, r_1, r_2	um dos registros, A, B, C, D, E, H ou L
DDD, SSS	uma configuração de bits para indicar um dos registros A, B, C, D, E, H ou L (DDD = destino, SSS = origem) DDD ou SSS podem ser: 111 — A 000 — B 001 — C 010 — D 011 — E 100 — H 101 — L
RP ou rp	Uma configuração de bits para representar os pares de registros B — representa BC D — representa DE H — representa HL SP — endereço de 16 bits que representa o Apontador da PILHA RP pode ser: 00 — BC 01 — DE 10 — HL 11 — SP

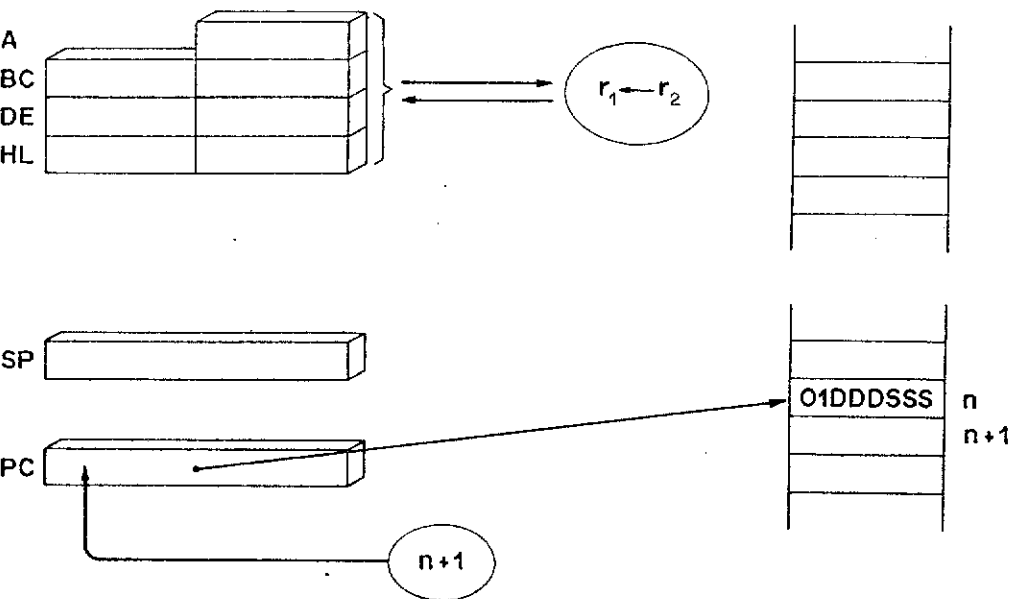
	Flags de condição:
Ac	Vai um auxiliar;
P	Paridade;
S	Sinal;
CY	Vai um;
Z	Zero;
( )	indica o conteúdo do registro ou de uma posição de memória, que se encontra entre parênteses.
$\vee$	*OU* lógico (OR)
$\wedge$	*E* lógico (AND)
$\veebar$	*OU EXCLUSIVO* lógico
|	concatenação

Para facilitar o estudo das instruções 8080/8085, vamos agrupá-las da seguinte forma:

1. Instruções de Transferência de Dados
2. Instruções Aritméticas
3. Instruções Lógicas
4. Instruções de Desvio
5. Instruções de Manipulação com a PILHA, E/S, Interrupções e outras.

1. INSTRUÇÕES DE TRANSFERÊNCIA DE DADOS

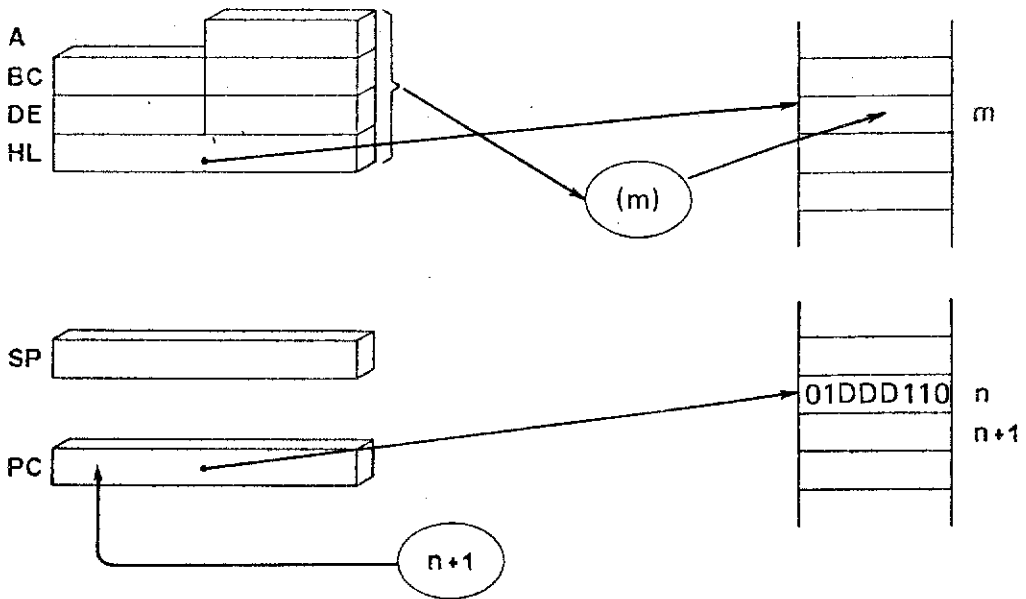
MOVIMENTAÇÃO ENTRE REGISTROS — MOV r_1, r_2

Consiste em mover o conteúdo de um registro para outro.

FLAGS: ______

Ex.:

MOV A,B	01111000	move o conteúdo do registro B para o acumulador
MOV B,B	01000000	não faz nada, visto que a origem e o destino são o mesmo registro

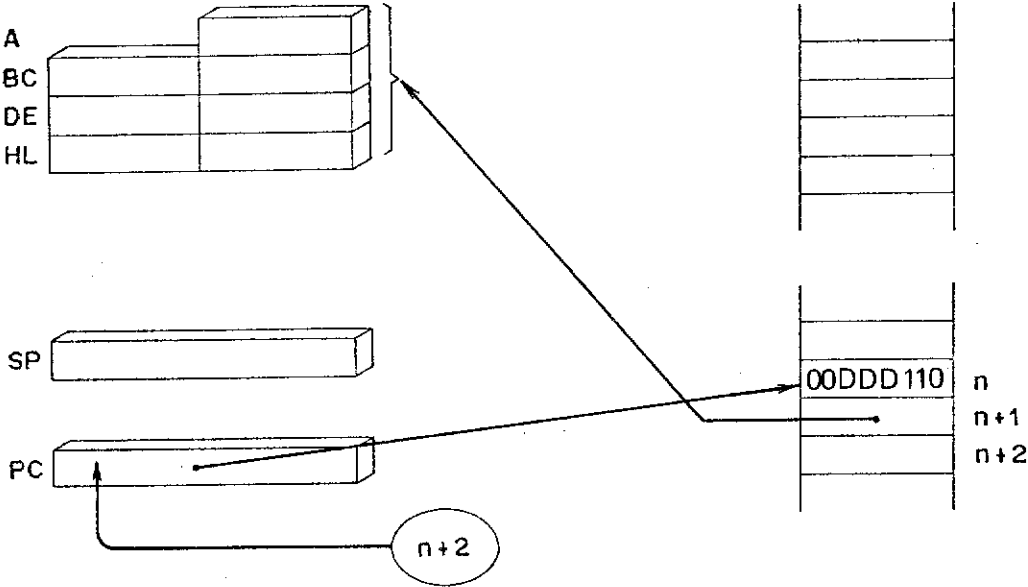
MOVIMENTAÇÃO ENTRE MEMÓRIA E UM REGISTRO — MOV r, m

Consiste em mover o conteúdo de uma posição de memória apontada pelo par de registro HL para um registro A, B, C, D, E, H ou L.

FLAGS: ______

Ex.:

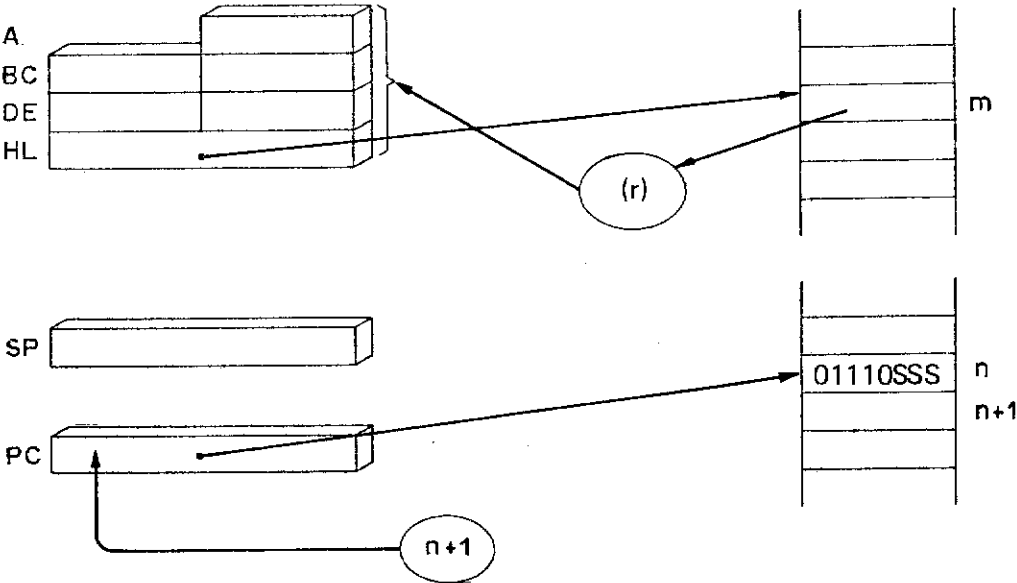
MOV A,M	01111110	move o conteúdo da memória apontada por HL para o acumulador
MOV L,M	01101110	move o conteúdo da memória apontada por HL para o registro L

MOVIMENTAÇÃO IMEDIATA PARA UM REGISTRO — MVI r, e

Consiste em mover o byte seguinte à instrução para um dos registros: A, B, C, D, E, H ou L.

FLAGS: ____

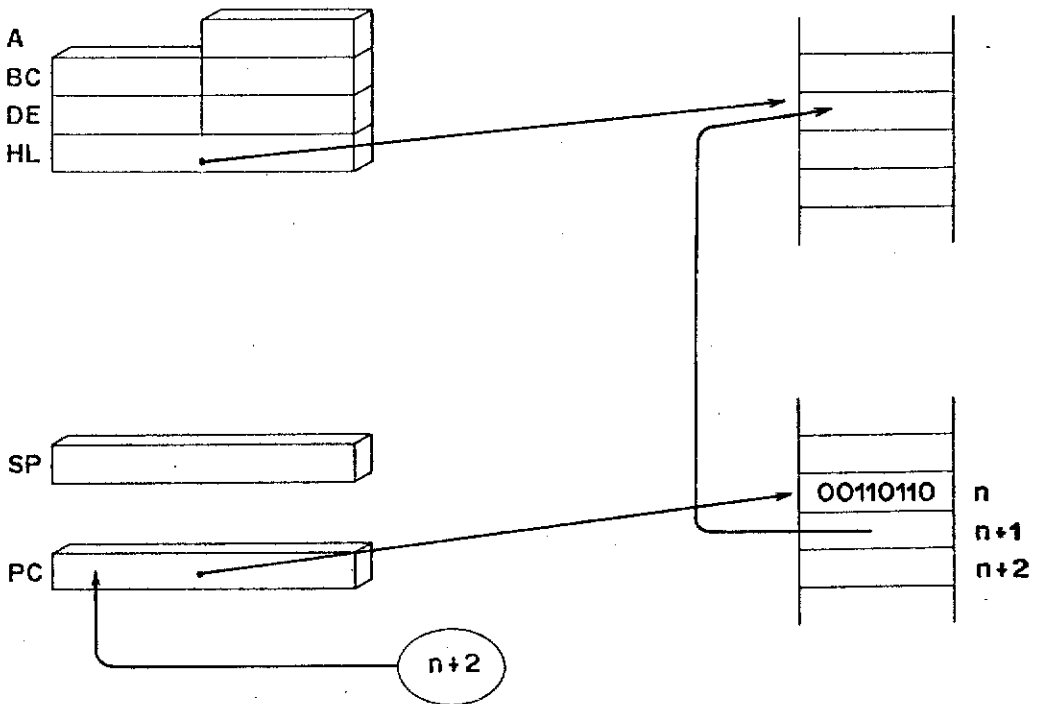
Ex.:
MVI A, 0FH   00111110 00001111   coloca no acumulador a constante $0F_{16}$

MOVIMENTAÇÃO ENTRE UM REGISTRO E A MEMÓRIA — MOV M, r

Consiste em mover o conteúdo de um registro: A, B, C, D, E, H ou L, para uma posição de memória apontada pelo par de registros HL.

FLAGS: ______

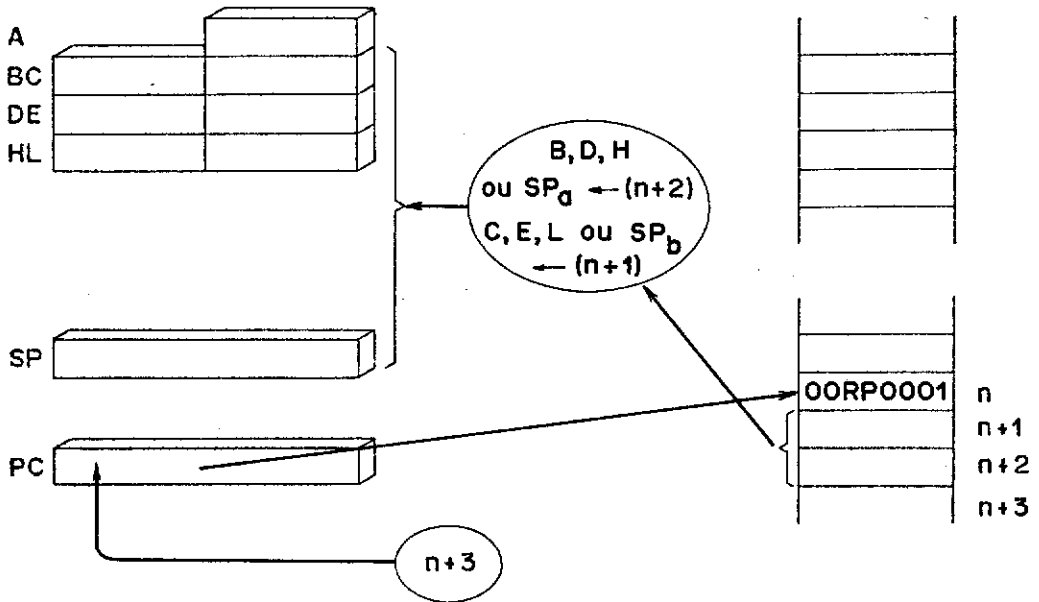
Ex.:
MOV M,C   01110001   move o conteúdo do registro C, para a posição de memória apontada por HL

MOVIMENTAÇÃO IMEDIATA PARA MEMÓRIA — MVI M, e

Consiste em mover o conteúdo da posição de memória seguinte à instrução para uma posição de memória cujo endereço se encontra no par de registros HL.

FLAGS: ______
Ex.:
MVI M,0   00110110 00000000   zera a posição de memória apontada por HL

CARGA IMEDIATA EM PAR DE REGISTRO — LXI rp, e16

Consiste em mover os 2 bytes seguintes à instrução para um dos pares de registros (BC, DE, HL ou SP). O primeiro byte é movido para o registro correspondente à parte baixa, e o segundo byte é movido para o registro correspondente à parte alta.

A
BC
DE
HL
SP
PC
B, D, H ou SP_a ← (n+2)
C, E, L ou SP_b ← (n+1)
00RP0001
n
n+1
n+2
n+3
n+3

FLAGS: _____

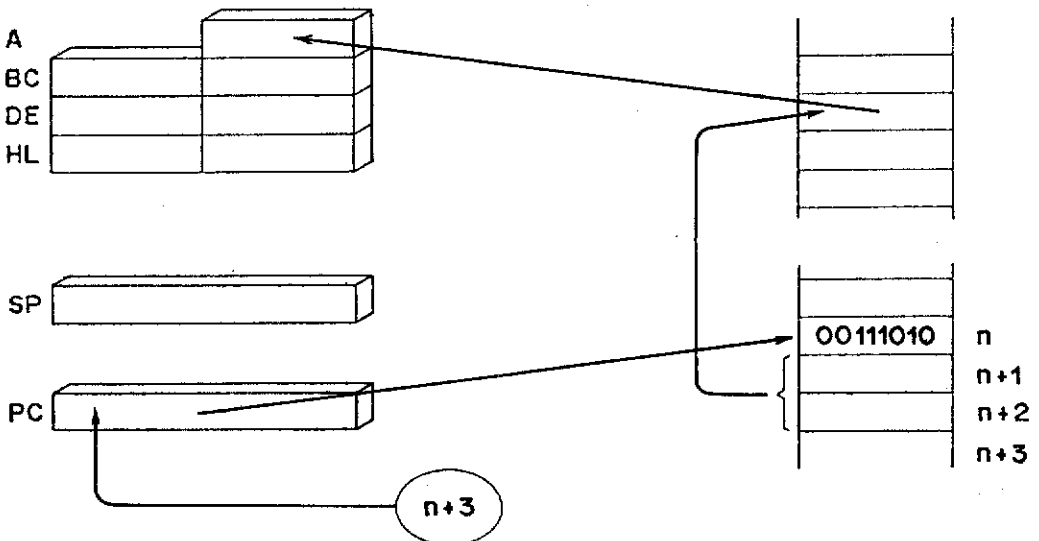
Ex.:

LXI B,100H 00000001 00000000 00000001 o par BC recebe 100_{16}

Observe que LXI B,100H é equivalente: MVI B,1
MVI C,0

LXI SP,4F0H 00110001 11110000 00000100 SP passa apontar para $4F0_{16}$

CARGA NO ACUMULADOR USANDO ENDEREÇAMENTO DIRETO — LDA end

Consiste em mover o conteúdo da posição de memória, cujo o endereço se encontra na posição seguinte ao código da instrução para o acumulador.

FLAGS: ____

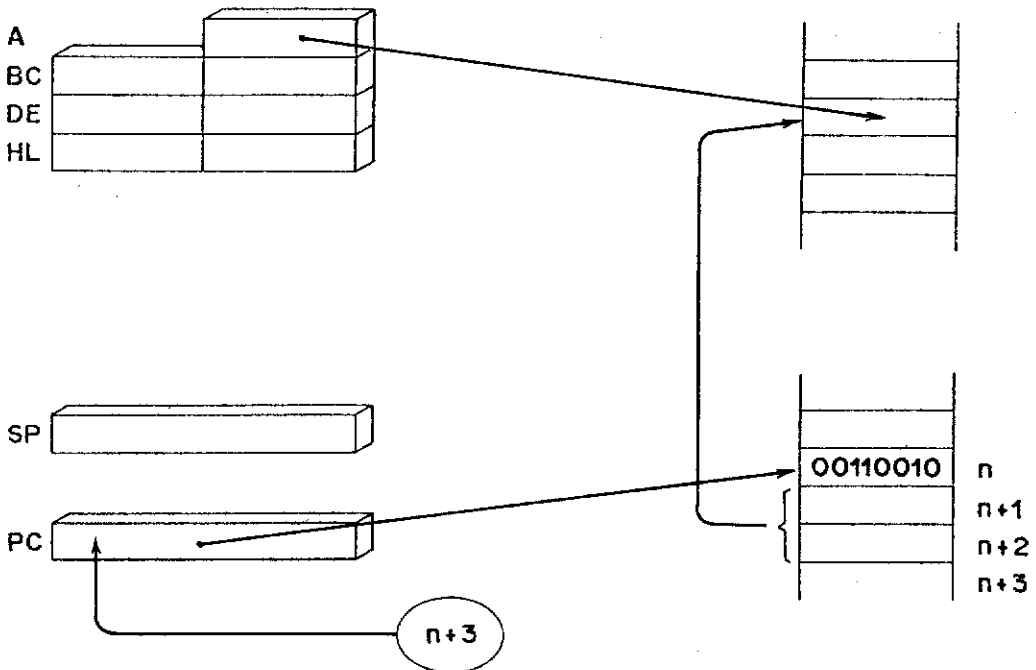
A posição de memória n + 1 contém a parte baixa do endereço e n + 2 a parte alta.

Ex.: Suponha que a posição de memória $01F0_{16}$ contenha $0D_{16}$.

LDA 01F0H 00111010 11110000 00000001 o acumulador conterá $0D_{16}$

Observe que esta instrução é equivalente a : LXI H, 01F0H
MOV A,M

ARMAZENAMENTO DO ACUMULADOR USANDO ENDEREÇO DIRETO — STA end

Consiste em mover o conteúdo do acumulador para uma posição de memória cujo endereço está especificado nos dois bytes seguintes ao código da instrução.

FLAGS: ____

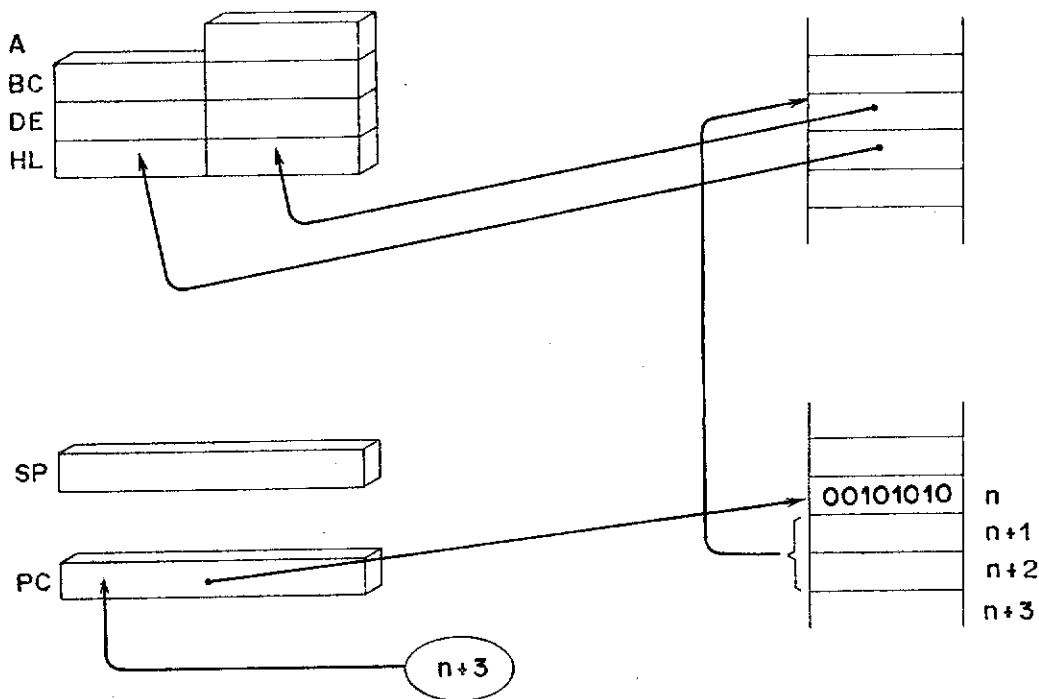
A posição de memória n + 1 contém a parte baixa do endereço, e n + 2 a parte alta.

Ex.: Suponha que o conteúdo do acumulador é $0E_{16}$.

STA 01F0H 00110010 11110000 00000001 a posição de memória $01F0_{16}$, passará a conter $0E_{16}$

Observe que esta instrução é equivalente a: LXI H,01F0H
MOV M,A

CARGA DIRETA PARA REGISTROS HL — LHLD end

Consiste em mover dois bytes consecutivos, que estão localizados em uma posição de memória indicada pelo endereço que segue a instrução, para os registros L e H, respectivamente.

FLAGS: ______

A posição de memória n + 1 contém a parte baixa do endereço, e n + 2, a parte alta.

Ex.: Suponha que 200_{16} e 201_{16} contém 0 e FF_{16} respectivamente.

LHLD 200H   00101010 00000000 00000010 o par HL passará a conter $FF00_{16}$

Esta instrução é útil quando o endereço de memória que HL indica varia freqüentemente durante o programa.

Observe se quiséssemos que HL recebesse sempre o mesmo valor poderíamos usar: LXI H, valor.

ARMAZENAMENTO DIRETO NA MEMÓRIA DOS REGISTROS H e L — SHLD end

Consiste em mover para dois bytes consecutivos de memória, que são indicados pelo endereço que segue a instrução, o conteúdo dos registros L e H respectivamente.

FLAGS: ———

A posição de memória n + 1 contém a parte baixa do endereço, e n + 2, a parte alta.

Ex.: Suponha que o par HL contém $FF00_{16}$

SHLD 200H   00100010 00000000 00000010   a posição de memória 200_{16} conterá 0 e a 201_{16} conterá FF_{16}

CARGA NO ACUMULADOR UTILIZANDO ENDEREÇO INDIRETO — LDAX rp

Consiste em mover o conteúdo de uma posição de memória para o acumulador. O endereço é fornecido por um dos pares de registros BC ou DE.

FLAGS: ——

Ex.: Suponha que D contém 10_{16} e E contém 00_{16}, e a posição de memória 1000_{16} contém FF_{16}.

LDAX D   00011010   o acumulador passará a conter FF_{16}

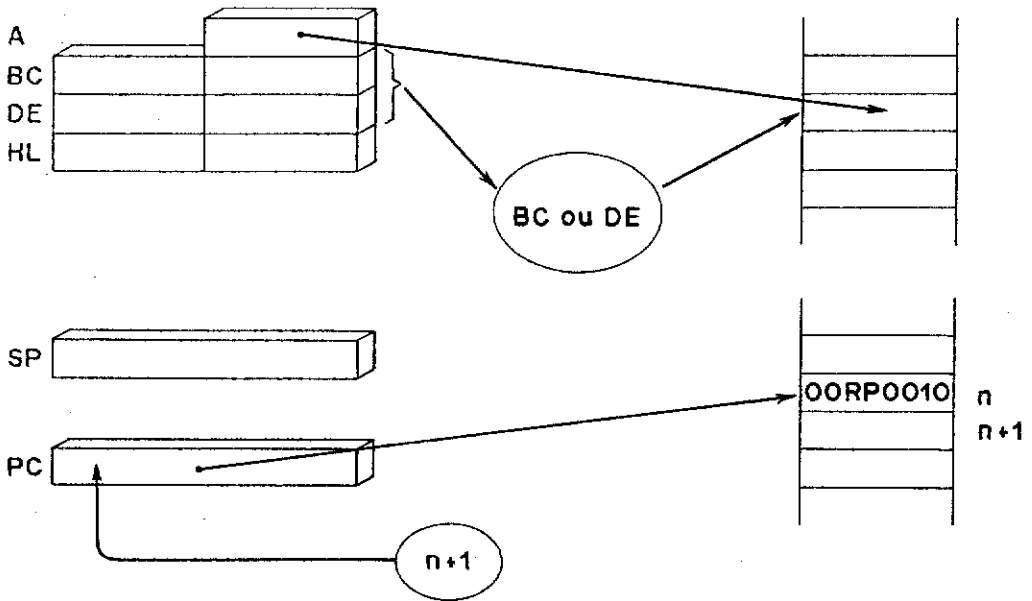
Com freqüência as instruções LDAX e LXI são utilizadas juntas de modo que LXI define o endereço e LDAX faz a transferência:

LXI D, 1000H

LDAX D

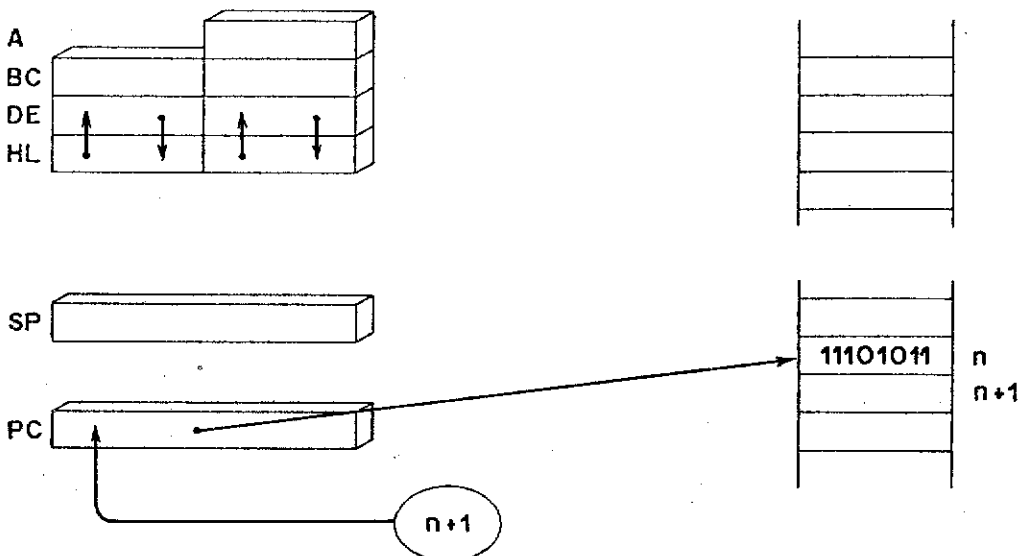
Observe que não existe a instrução LDAX H, pois existe a instrução MOV A,M.

ARMAZENAMENTO DO ACUMULADOR UTILIZANDO ENDEREÇO INDIRETO — STAX rp

Consiste em mover o conteúdo do acumulador para uma posição de memória, cujo endereço é fornecido por um dos pares de registros BC ou DE.

FLAGS: ______

Ex.: Suponha que B contém $0F_{16}$ e C contém 08_{16}, e o acumulador FF_{16}.
STAX B 00000010 a posição de memória $0F08_{16}$ passará a conter FF_{16}

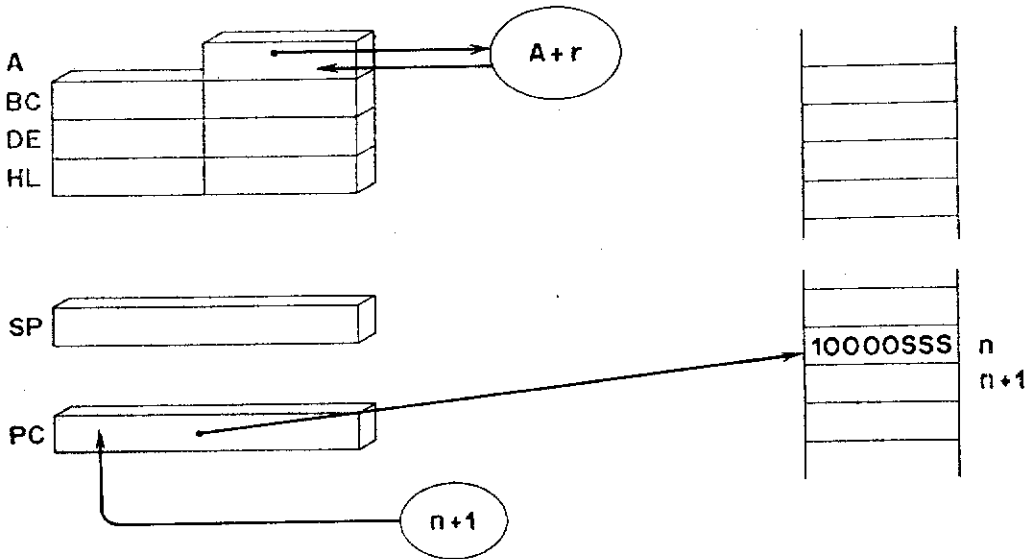
TROCA DE PAR DE REGISTROS (DE com HL) — XCHG

Consiste em trocar o conteúdo dos registros: D com H e E com L.

FLAGS: ______

Ex.: Suponha os seguintes conteúdos dos registros:
D = 10_{16}, E = 00_{16}, H = 30_{16}, L = $A0_{16}$
XCHG   11101011   DE passará a conter: $30A0_{16}$ e HL conterá: 1000_{16}

2. INSTRUÇÕES ARITMÉTICAS

SOMA COM REGISTRO — ADD r

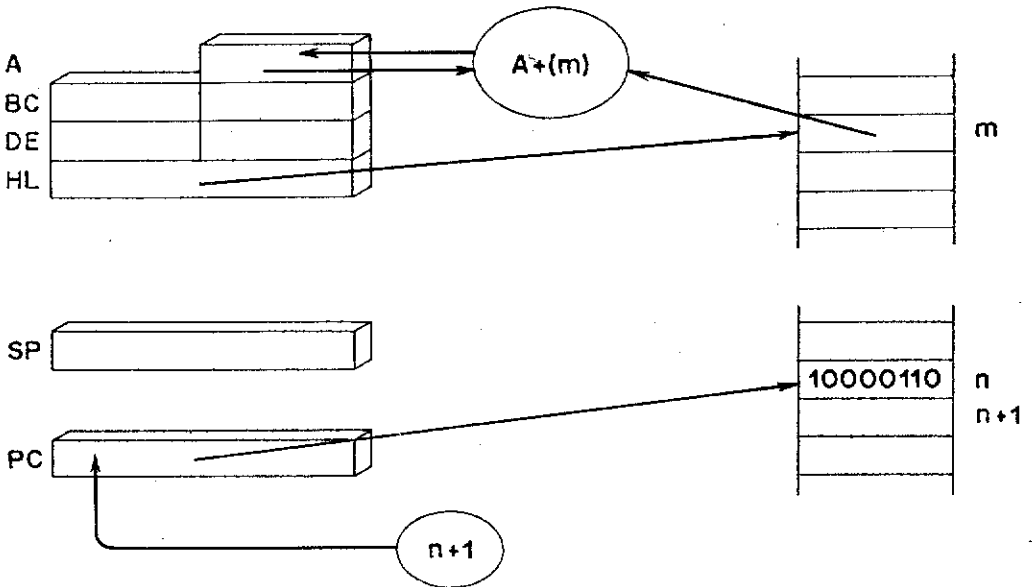
Consiste em somar o conteúdo de um dos registros: A, B, C, D, E, H ou L com o acumulador.

FLAGS: A_C, P, S, CY e Z

Ex.: Suponha que B = $F3_{16}$ e A = $B0_{16}$
ADD B   10000000   o acumulador conterá $A3_{16}$

$$\begin{array}{r} 11110011 \\ +\ 10110000 \\ \hline 10100011 \end{array} \Longrightarrow A3_{16}$$

CY = 1
S = 1
$A_c = 0$
P = 1 (nº par de bits 1)
Z = 0

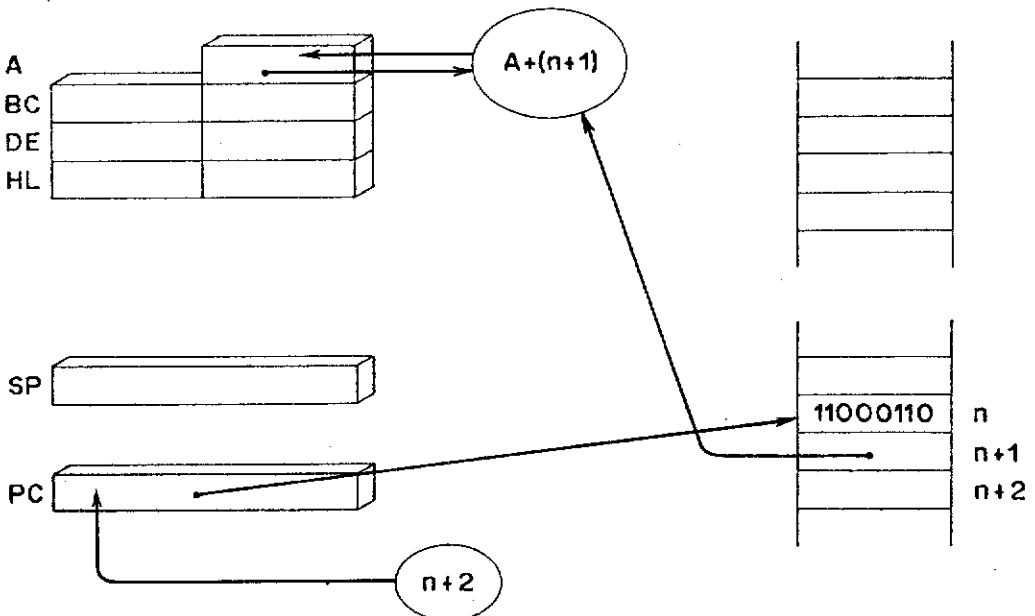
SOMA COM A MEMÓRIA — ADD M

Consiste em somar o conteúdo do acumulador com o de uma posição de memória, cujo endereço se encontra no par de registros HL.

FLAGS: A_c, P, S, CY e Z

Ex.: Suponha os seguintes conteúdos: $A = 09_{16}$, $HL = 01A7_{16}$ e $(01A7) = C8_{16}$
ADD M 10000110 o acumulador conterá: $D1_{16}$
$A_c = 1$, $P = 1$, $S = 1$; $CY = 0$, $Z = 0$

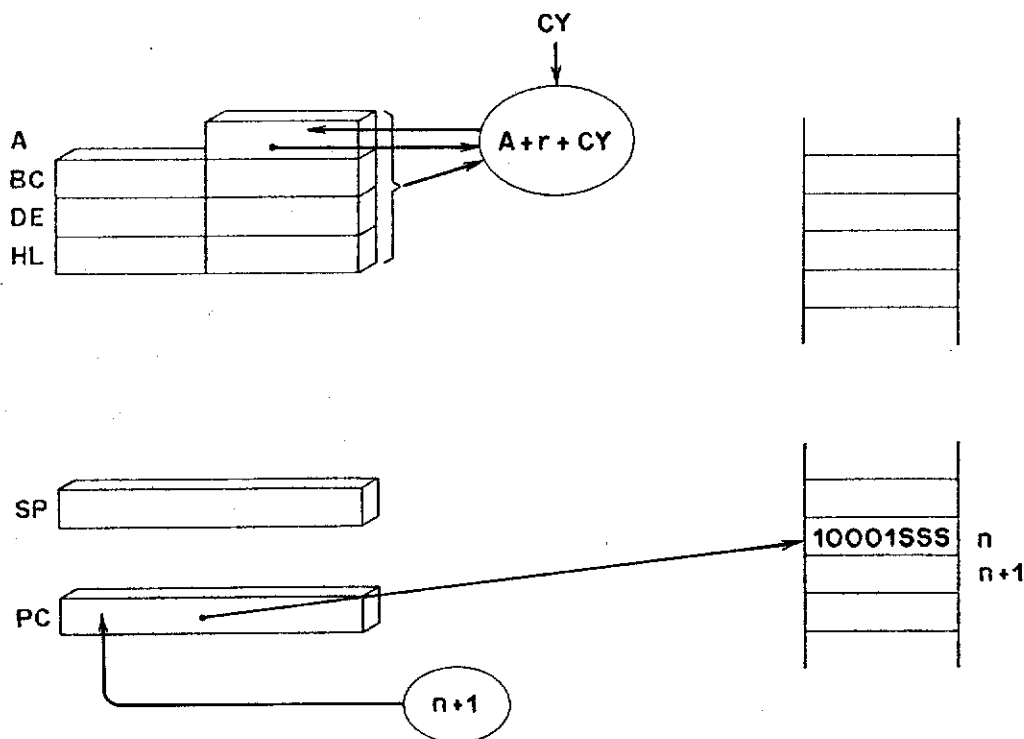
SOMA IMEDIATA — ADI e

Consiste em somar o byte seguinte à instrução com o acumulador.

A
BC
DE
HL
A+(n+1)
SP
PC
11000110
n
n+1
n+2
n+2

FLAGS: A_c, P, S, CY e Z

Ex.: Suponha que A = $7C_{16}$
ADI 3AH 11000110 00111010 o acumulador conterá: $B6_{16}$
$A_c = 1$, $P = 0$, $S = 1$, $CY = 0$, $Z = 0$

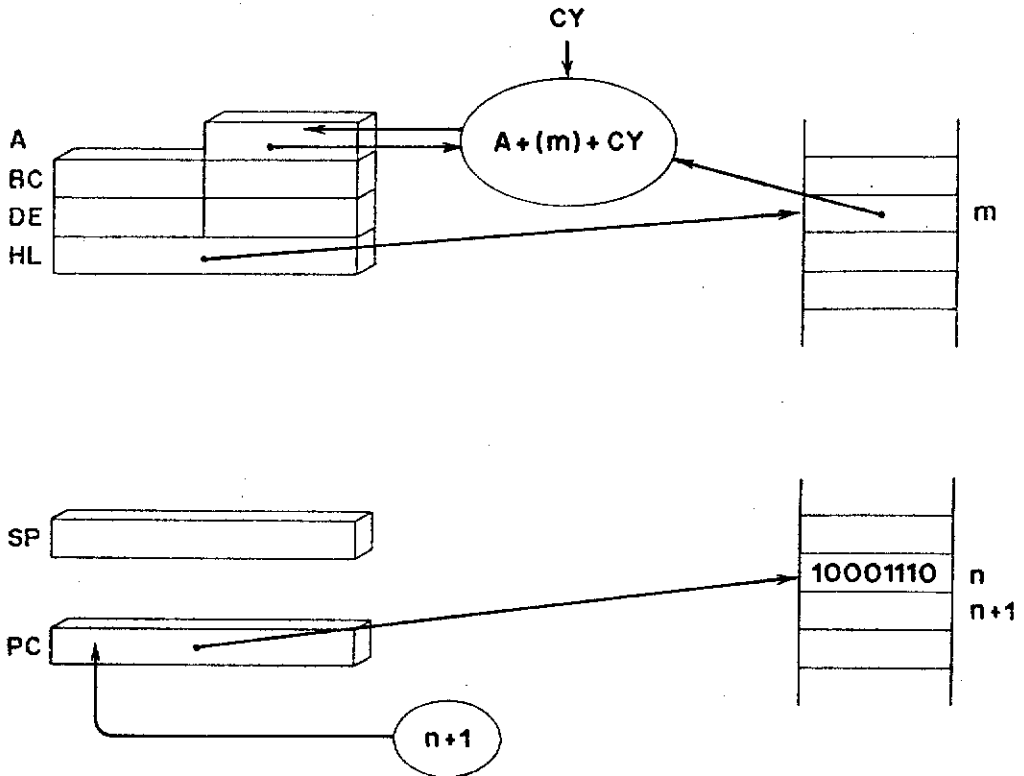
SOMA COM REGISTRO E VAI UM — ADC r

Consiste em somar o conteúdo de um dos registros A, B, C, D, E, H ou L com o acumulador e mais o flag VAI UM.

FLAGS: A_c, P, S, CY e Z

Ex.: Suponha que A = $D7_{16}$, E = $5A_{16}$, CY = 1
ADC E 10001011 o acumulador conterá 32_{16}

```
              11010111 (A)
              01011010 (E)
                     1 (CY)
              --------
CY = 1 ←----- 00110010
      S = 0 --┘    U        Ac = 1        P = 0
                                          Z = 0
```

SOMA COM MEMÓRIA E VAI UM — ADC M

Consiste em somar o conteúdo do acumulador com o flag VAI UM e uma posição de memória, cujo endereço se encontra no par de registros HL.

FLAGS: A_c, P, S, CY e Z

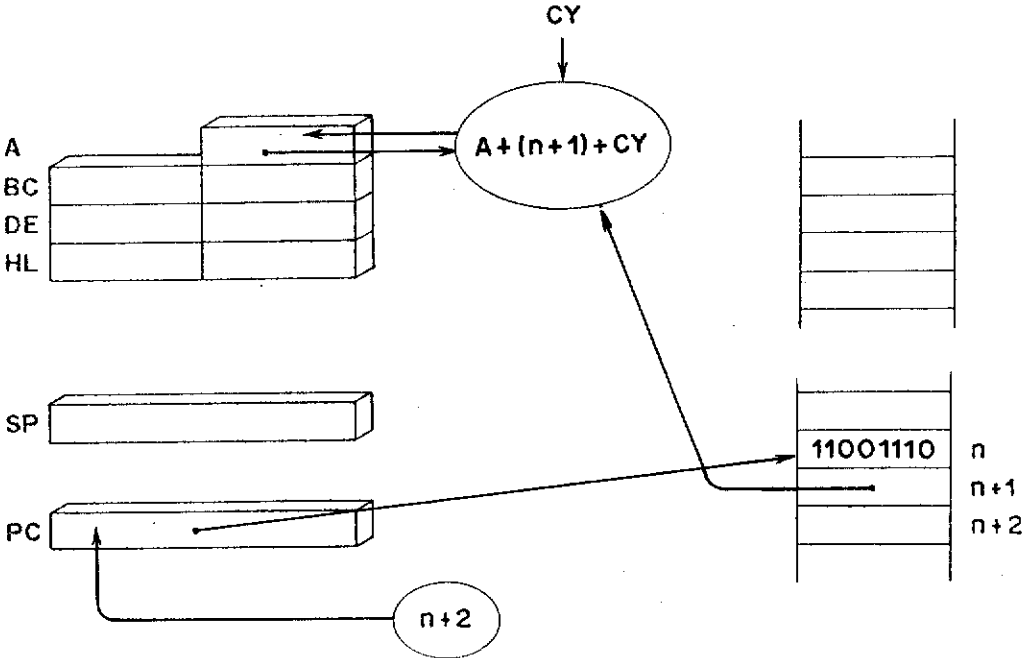
Ex.: Suponha que A = $D7_{16}$, HL = 100_{16}, (100_{16}) = 3 e CY = 0

ADC M   10001110   o acumulador conterá DA_{16}

$A_c = 0$, P = 0, S = 1, CY = 0, Z = 0

SOMA IMEDIATA COM VAI UM — ACI e

Consiste em somar o acumulador com o flag VAI UM e com o byte seguinte à instrução.

CY
A
BC
DE
HL
A + (n + 1) + CY
SP
PC
11001110 n
n + 1
n + 2
n + 2

FLAGS: A_c, P, S, CY e Z

Ex.: Suponha que A = $9A_{16}$, (n + 1) = $3B_{16}$, CY = 1
ACI 3BH 11001110 00111011 o acumulador conterá: $D6_{16}$
A_c = 1, P = 0, S = 1, CY = 0 e Z = 0

SUBTRAÇÃO COM REGISTRO — SUB r

Consiste em subtrair o conteúdo de um dos registros: A, B, C, D, E, H ou L do acumulador.

FLAGS: A_c, P, S, CY e Z

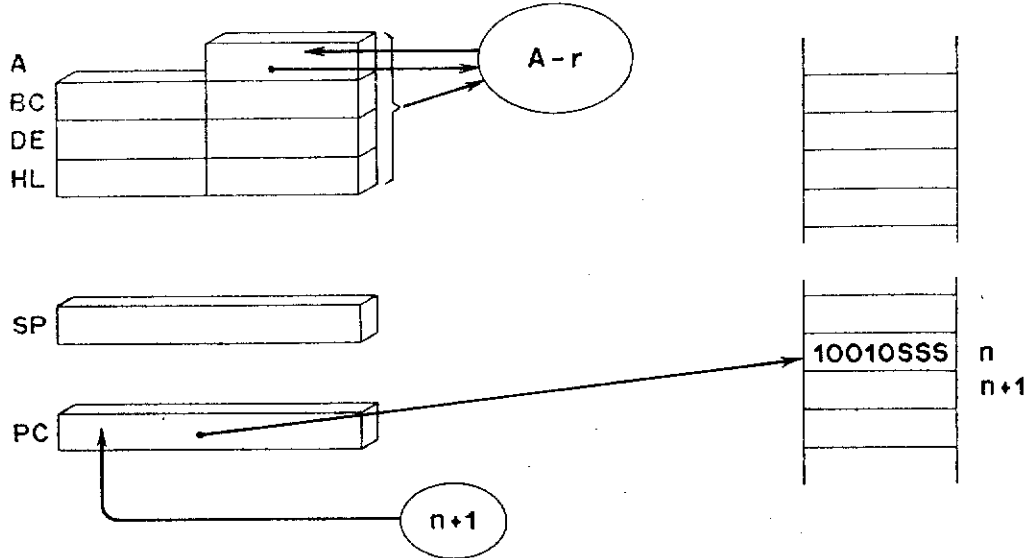
Ex.: Suponha que A = $F3_{16}$ e B = $B0_{16}$
SUB B 10010000 o acumulador conterá 43_{16}

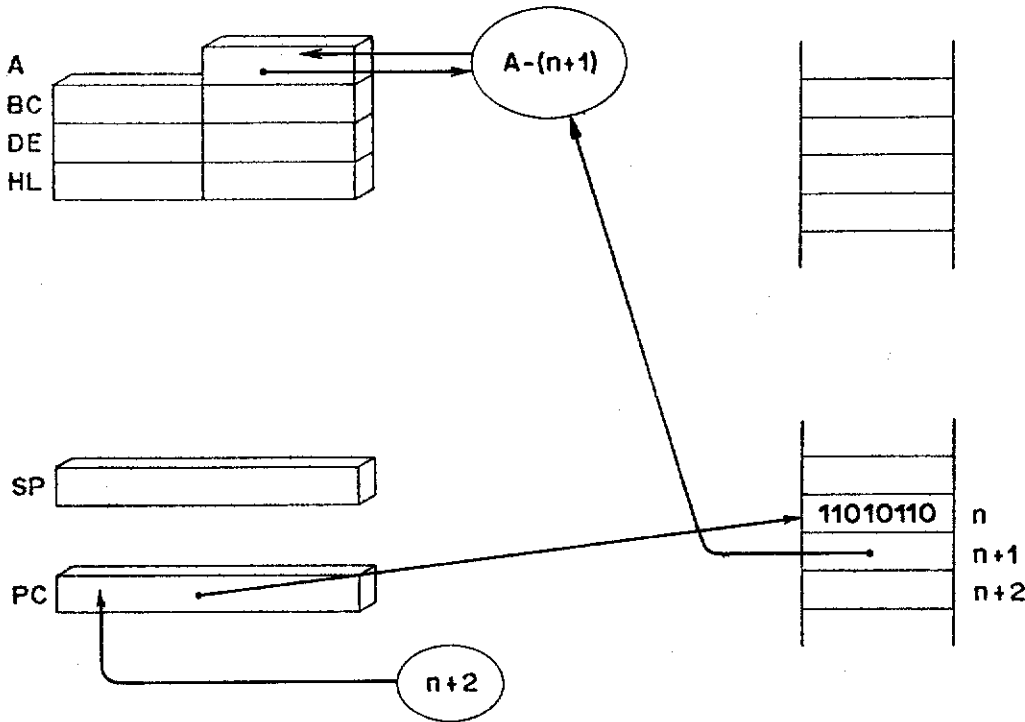
Observe que o CY é complementado.

SUBTRAÇÃO COM MEMÓRIA — SUB M

Consiste em subtrair do acumulador o conteúdo de uma posição de memória, cujo endereço se encontra no par de registros HL.

FLAGS: A_c, P, S, CY e Z

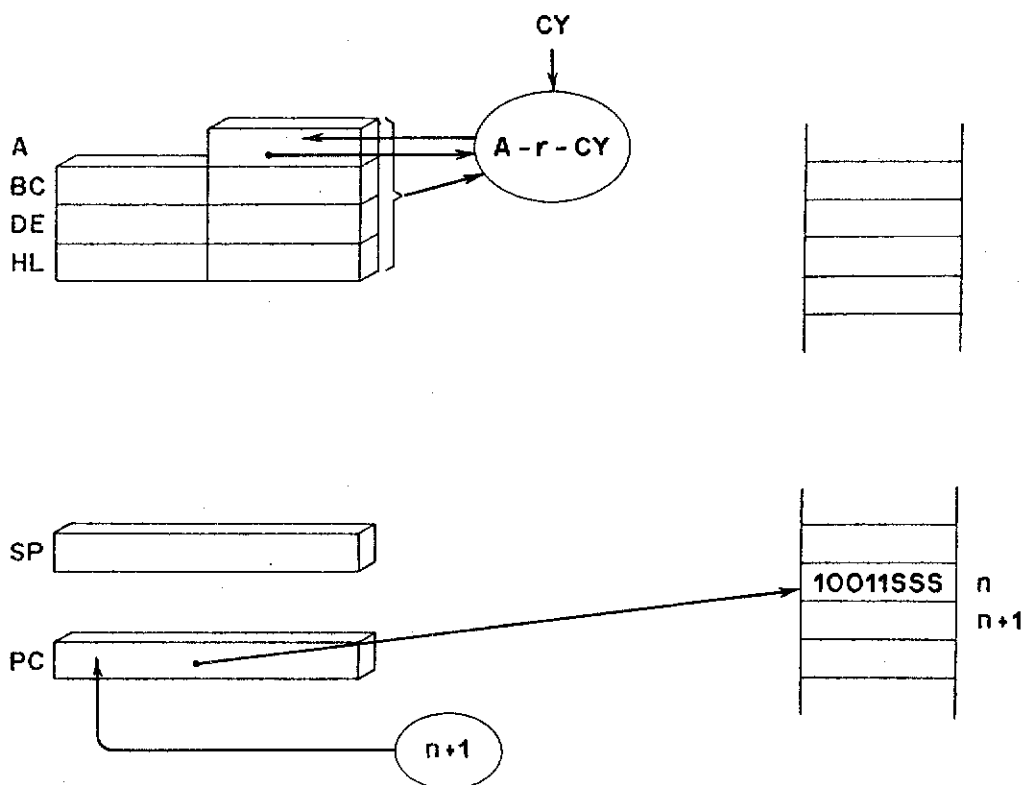
Ex.: Suponha os seguintes conteúdos: A = 09_{16}, HL = $01A7_{16}$ e (01A7) = $C8_{16}$
SUB M 10010110 o acumulador conterá: 41_{16}
A_c = 0, P = 1, S = 0, CY = 1, Z = 0

SUBTRAÇÃO IMEDIATA — SUI e

Consiste em subtrair do acumulador o conteúdo do byte seguinte à instrução.

FLAGS: A_c, P, S, CY e Z

Ex.: Suponha que A = $7C_{16}$

SUI 3AH 11010110 00111010 o acumulador conterá: 42_{16}
$A_c = 0$, $P = 1$, $S = 0$, $CY = 0$, $Z = 0$

SUBTRAÇÃO COM REGISTRO E VAI UM — SBB r

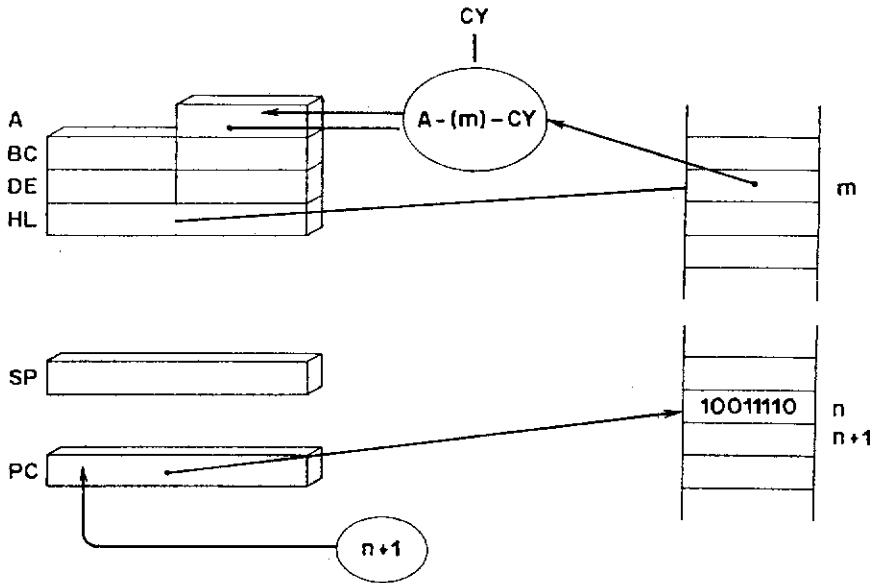
Consiste em subtrair do acumulador o conteúdo de um dos registros A, B, C, D, E, H ou L e o VAI UM.

FLAGS: A_c, P, S, CY e Z

Ex.: Suponha A = $D7_{16}$, E = $5A_{16}$, CY = 1

SBB E   10011011   o acumulador conterá $7C_{16}$

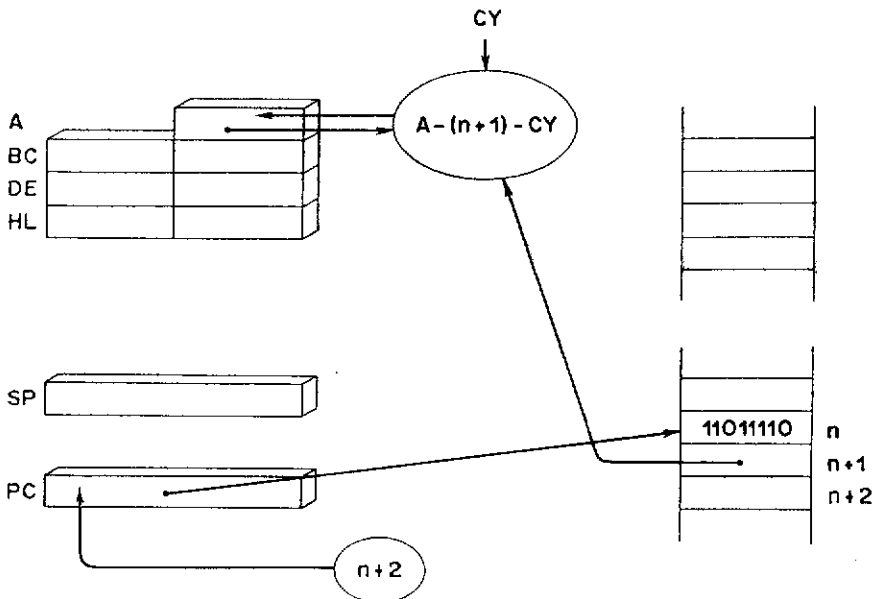
$D7_{16}$ ——————→ 11010111
Comp. a 2 de $5A_{16}$ ——→ + 10100110
Comp. a 2 do 1   11111111
┌— 1 ——— 01111100
CY = 0   ↑
S = 0 $A_c = 1$ P = 0
Z = 0

SUBTRAÇÃO COM MEMÓRIA E VAI UM — SBB M

Consiste em subtrair do acumulador o conteúdo de uma posição de memória, cujo endereço se encontra no par de registros HL, e o flag VAI UM.

FLAGS: A_c, P, S, CY e Z

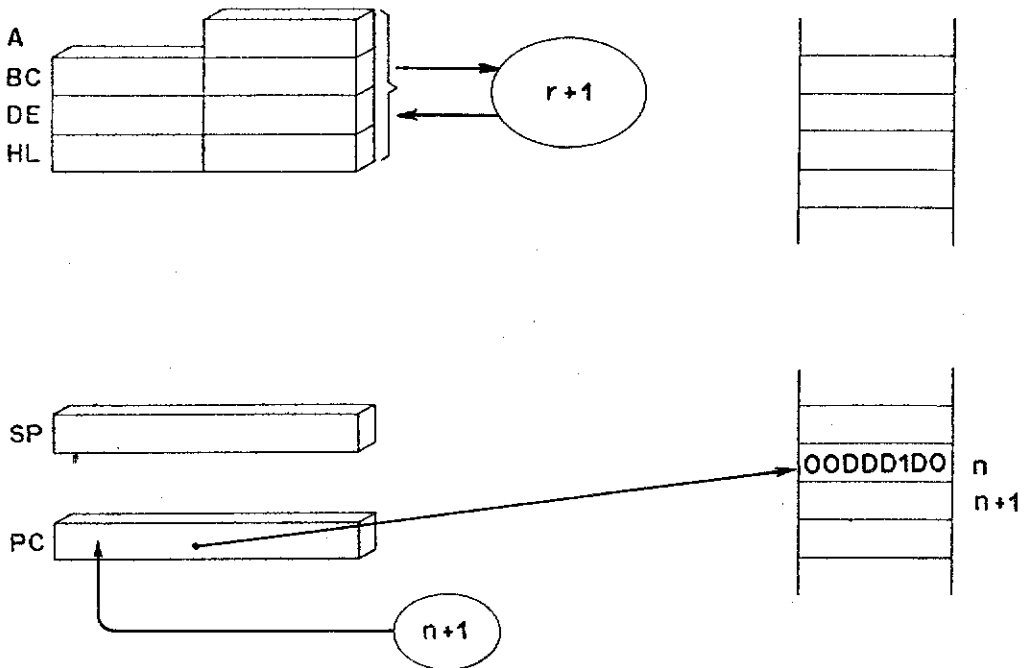
Ex.: Suponha que A = $D7_{16}$, HL = 100_{16}, (100_{16}) = 3_{16}, CY = 0
SBB M   10011110   o acumulador conterá $D4_{16}$
A_c = 1, P = 1, S = 1, CY = 0, Z = 0

SUBTRAÇÃO IMEDIATA COM VAI UM — SBI e

Consiste em subtrair do acumulador o byte seguinte à instrução e o flag VAI UM.

FLAGS: A_c, P, S, CY e Z

Ex.: Suponha que A = $9A_{16}$, (n + 1) = $3B_{16}$, CY = 1
SBI 3BH 11011110 00111011 o acumulador conterá $5E_{16}$
A_c = 1, P = 0, S = 0, CY = 0 e Z = 0

INCREMENTA REGISTRO — INR r

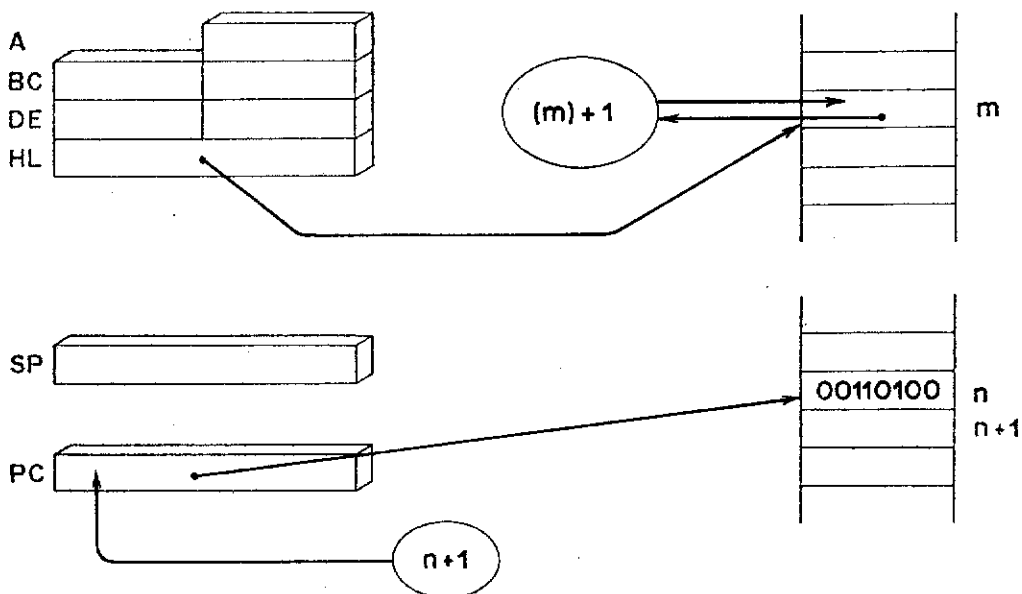
Consiste em somar 1, a um dos registros A, B, C, D, E, H ou L.

FLAGS: A_c, P, S e Z

Ex.: Suponha A = $3A_{16}$
INR A 00111100 o acumulador conterá $3B_{16}$
A_c = 0, P = 0, S = 0, Z = 0
Suponha B = FF_{16}
INR B 00000100 o registro B conterá 0
A_c = 1, P = 1, S = 0, Z = 1

INCREMENTA POSIÇÃO DE MEMÓRIA — INR M

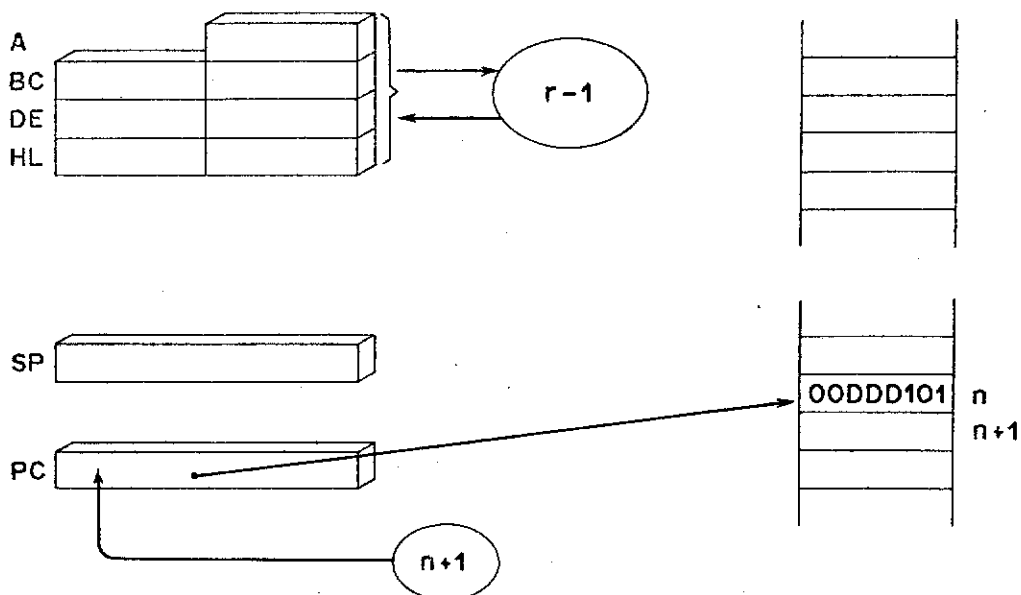
Consiste em somar 1 ao conteúdo de uma posição de memória cujo endereço se encontra no par de registros HL.

A
BC
DE
HL
(m) + 1
m

SP
00110100
n
n + 1
PC
n + 1

FLAGS: A_c, P, S e Z

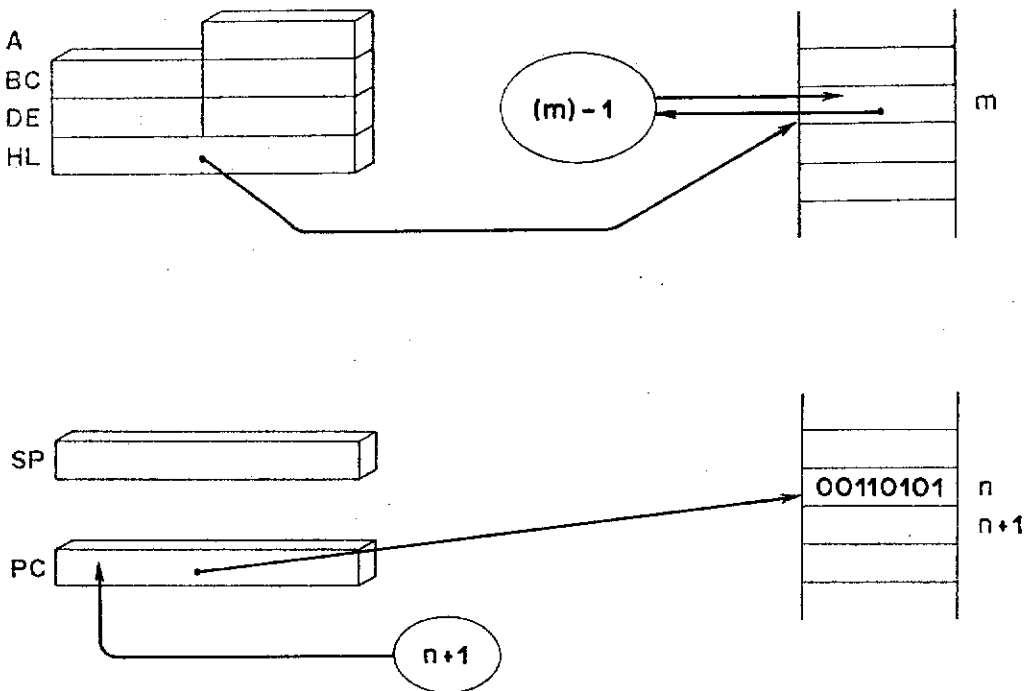
Ex.: Suponha HL = $5B7_{16}$ e ($5B7_{16}$) = $5F_{16}$
INR M   00110100   o conteúdo de posição $5B7_{16}$ será 60_{16}
$A_c = 1$, $P = 1$, $S = 0$, $Z = 0$

DECREMENTA REGISTRO — DCR r

Consiste em subtrair 1, de um dos registros A, B, C, D, E, H ou L.

FLAGS: A_c, P, S e Z

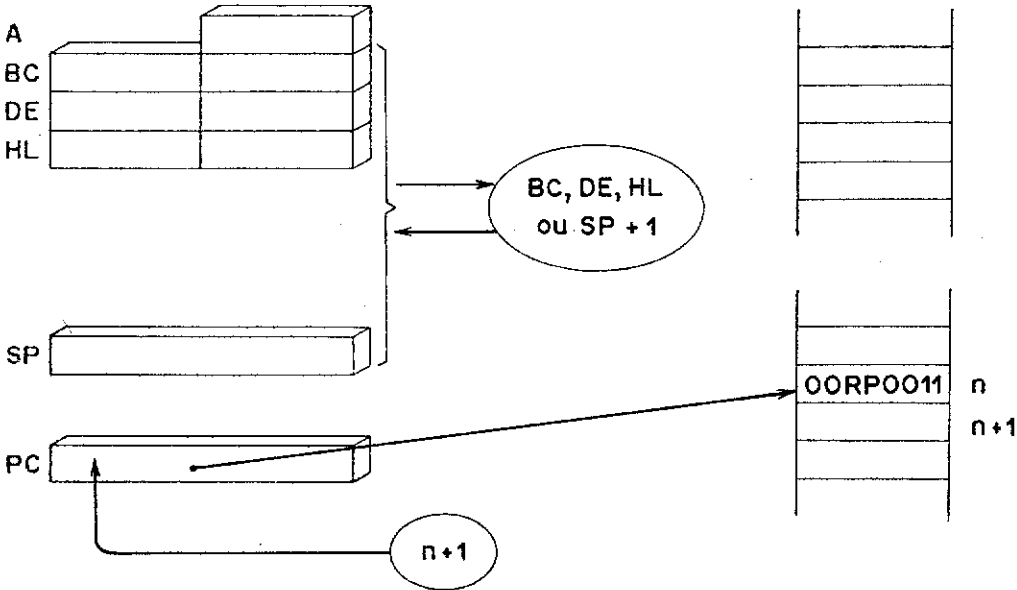
Ex.: Suponha D = 0
DCR D   00010101   o registro D conterá FF_{16}
$A_c = 1, P = 1, S = 1, Z = 0$

DECREMENTA POSIÇÃO DE MEMÓRIA — DCR M

Consiste em subtrair 1 do conteúdo de uma posição de memória cujo endereço se encontra no par de registros HL.

FLAGS: A_c, P, S e Z

Ex.: Suponha HL = $5B7_{16}$ e $(5B7_{16})$ = $5F_{16}$
DCR M   00110101   o conteúdo de posição $5B7_{16}$ será $5E_{16}$
$A_c = 0, P = 0, S = 0, Z = 0$

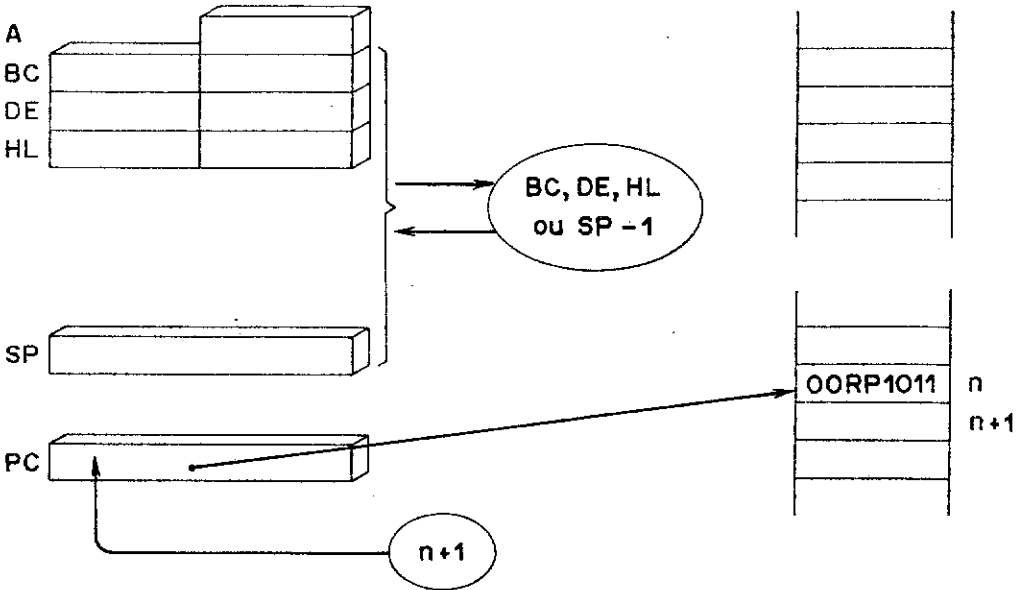
INCREMENTA PAR DE REGISTROS — INX rp

Consiste em somar 1 ao conteúdo de um dos pares de registros BC, DE, HL ou do Apontador de Pilha (SP).

FLAGS: ____

Ex.: Suponha BC = $3D6B_{16}$
INX B   00000011   o par BC conterá $3D6C_{16}$

DECREMENTA PAR DE REGISTROS — DCX rp

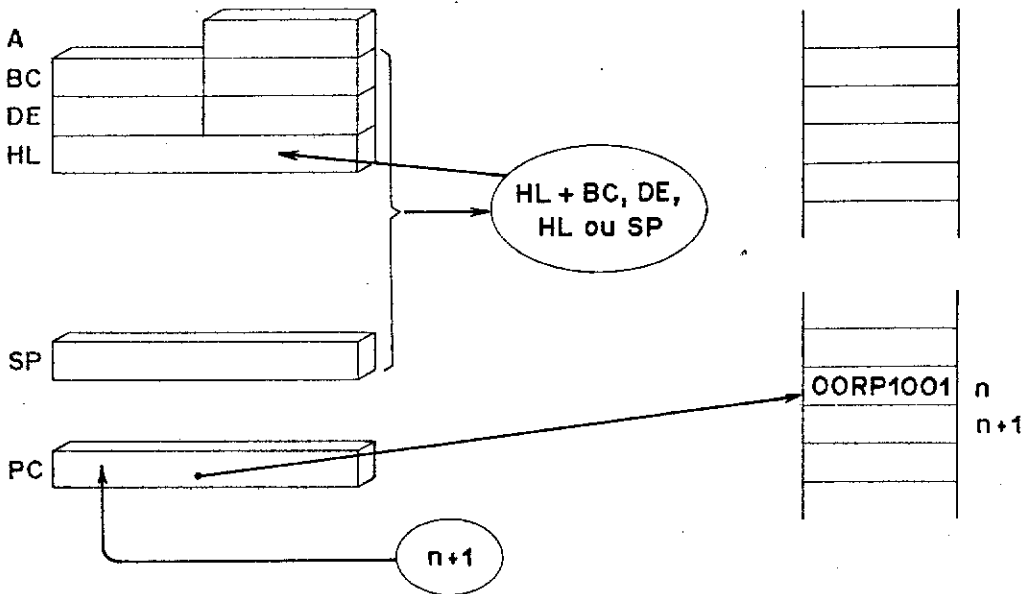
Consiste em subtrair 1 do conteúdo de um dos pares de registros BC, DE, HL ou do Apontador de Pilha (SP).

A
BC
DE
HL
BC, DE, HL
ou SP − 1
SP
00RP1011
n
n+1
PC
n+1

FLAGS: ——

Ex.: Suponha BC = $3D6B_{16}$
DCX B   00001011   o par BC conterá $3D6A_{16}$

A instrução DCX pode ser usada para controlar "loops", quando é necessário um contador maior que 256, neste caso porém é preciso acrescentar outras instruções para controlar o fim do "loop", visto que os flags não são alterados.

SOMA PAR DE REGISTRO COM HL — DAD rp

Consiste em somar o valor de 16 bits que se encontra em um dos pares de registros BC, DE, HL ou o Apontador de Pilha (SP) ao conteúdo do par HL.

FLAGS: CY

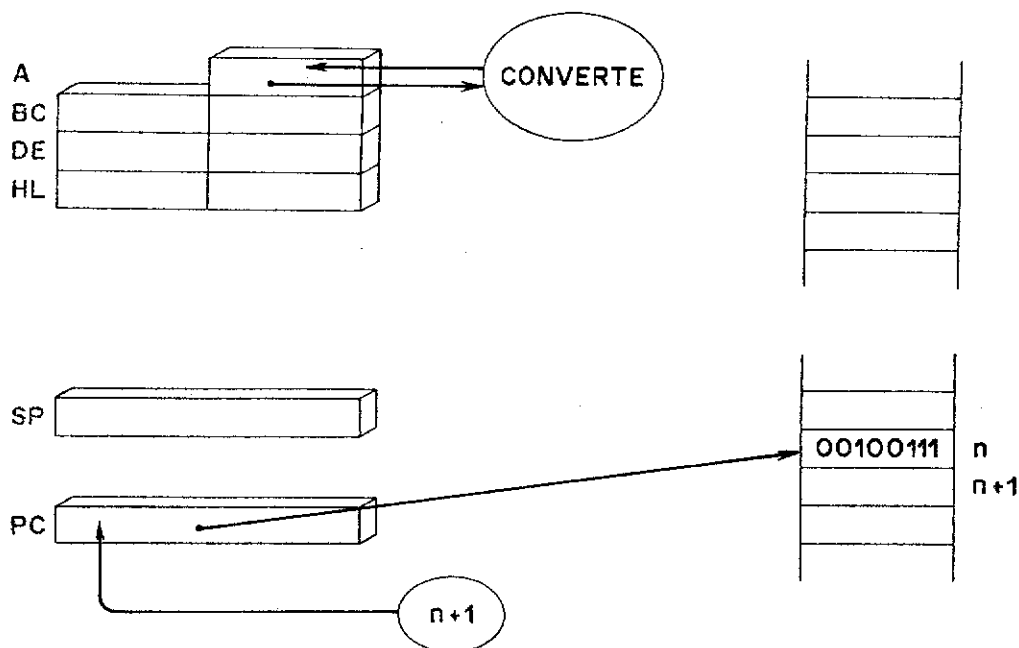
Ex.: Suponha HL = $057B_{16}$ e DE = $274A_{16}$
DAD D   00011001   HL conterá $2CC5_{16}$
CY = 0

AJUSTE DECIMAL — DAA

Consiste em ajustar o acumulador ao código BCD (Binary Coded Decimal). Esta instrução é somente usada após a soma de 2 números em BCD, garantindo que o resultado da soma permaneça em BCD.

O acumulador é ajustado da seguinte forma:

1. Se o valor dos 4 bits de menor significância do acumulador é maior que 9 ou se o flag VAI UM AUXILIAR está ligado, é somado 6 ao acumulador.

2. Se o valor dos 4 bits de maior significância do acumulador é agora maior que 9, ou se o flag VAI UM está ligado, 6 é somado aos 4 bits de maior significância do acumulador.

FLAGS: A_c, P, S, CY e Z

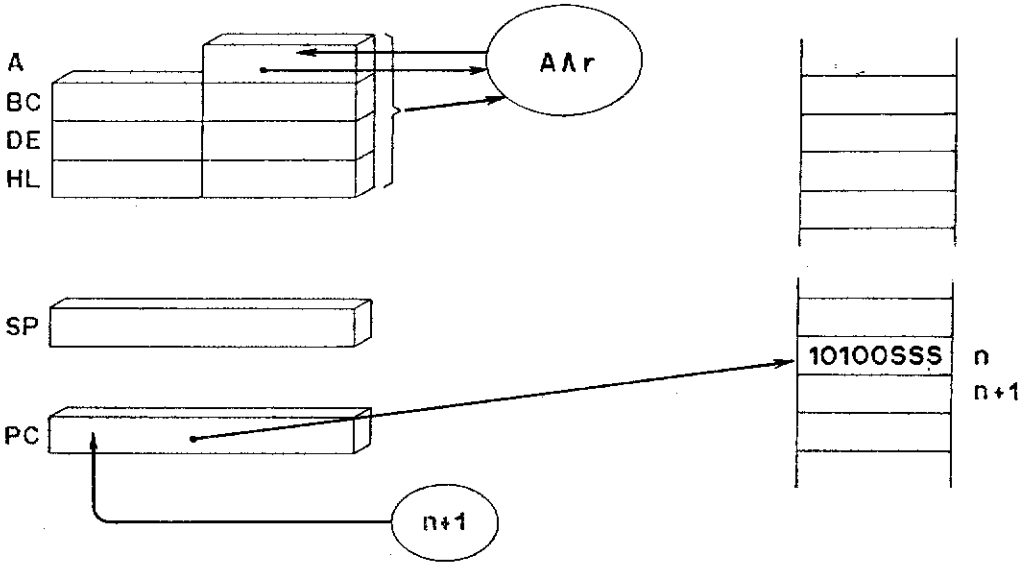
Ex.: Suponha que A = 38_{16} e C = 48_{16}
ADD C 10000001
DAA 00100111 o acumulador conterá 86_{16}
CY = 0

$$
\begin{array}{r} 00111000 \\ +\ 01001000 \\ \hline 10000000 \end{array} + 00000110 = 10000110
$$

A instrução DAA altera todos os flags, mas somente o VAI UM tem significado.

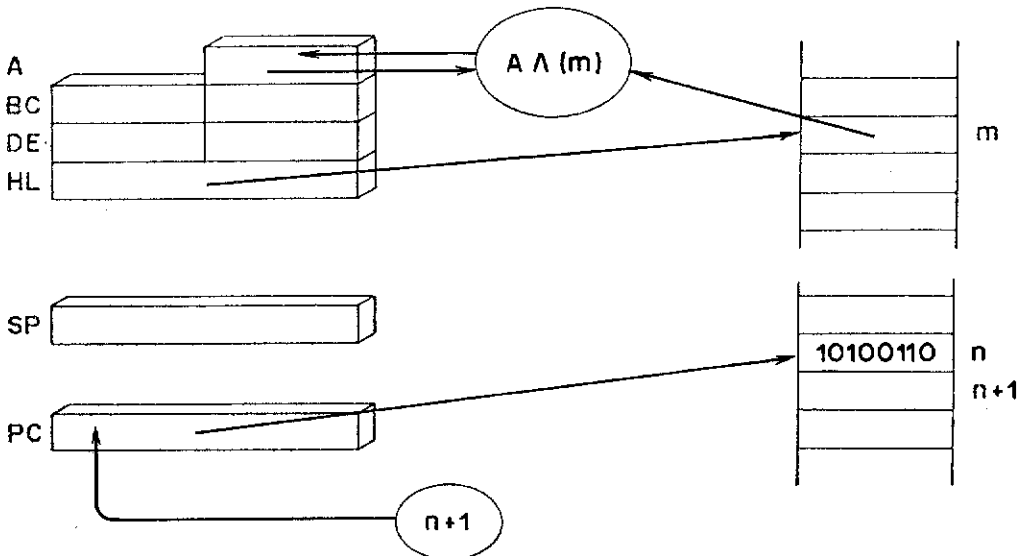
3. INSTRUÇÕES LÓGICAS

"E" COM REGISTRO — ANA r

Consiste em uma operação booleana e entre o conteúdo de um dos registros A, B, C, D, E, H ou L e o acumulador.

FLAGS: P, S e Z
$CY = 0$, $A_c = 1$

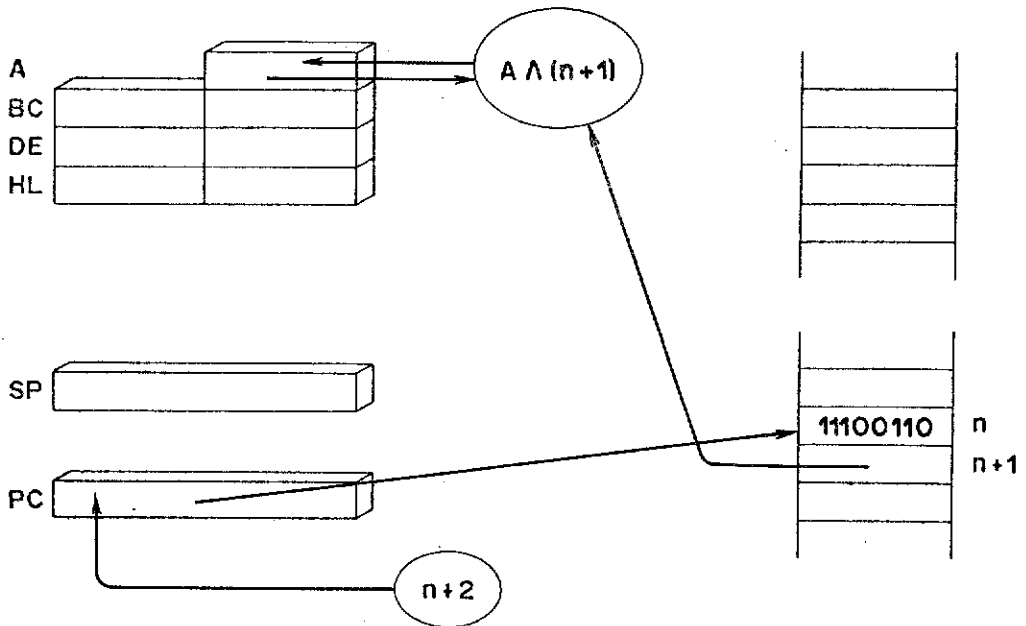
Ex.: Suponha $A = E3_{16}$ e $C = A0_{16}$
ANA C 10100001 o acumulador conterá $A0_{16}$
11100011 ∧ 10100000 → 10100000
$S = 1$, $P = 1$, $Z = 0$, $CY = 0$, $A_c = 1$

"E" COM POSIÇÃO DE MEMÓRIA — ANA M

Consiste em uma operação booleana *e* entre o conteúdo de uma posição de memória, cujo endereço se encontra no par de registros HL e o acumulador.

FLAGS: P, S e Z
$CY = 0, A_c = 1$

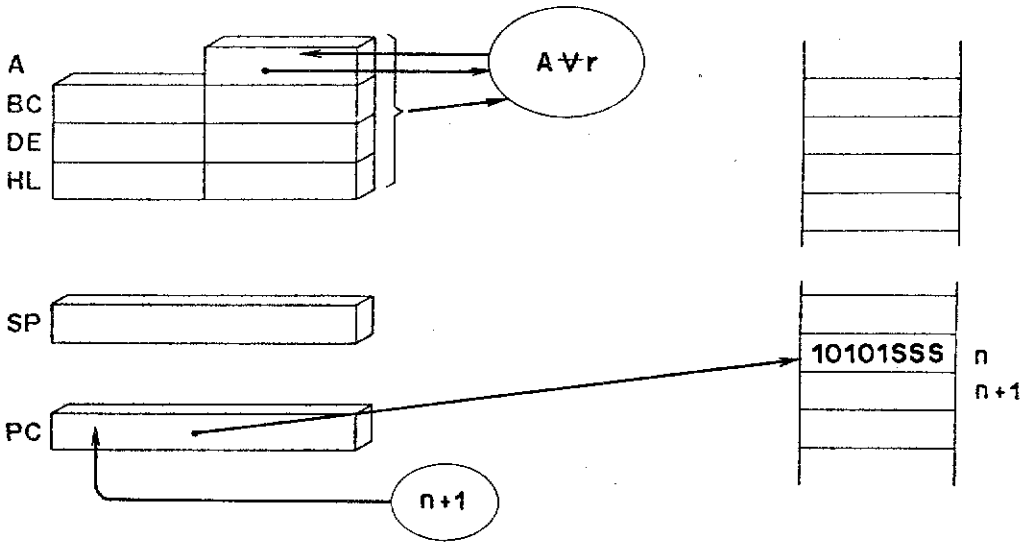
Ex.: Suponha $A = E3_{16}$, $HL = 3A53_{16}$,$(3A53_{16}) = 5F_{16}$
ANA M 10100110 o acumulador conterá 43_{16}
11100011 ∧ 01011111 → 01000011
$A_c = 1, CY = 0, P = 0, S = 0, Z = 0$

"E" IMEDIATO — ANI e

Consiste em uma operação booleana *e* entre o byte seguinte à instrução e o conteúdo do acumulador.

FLAGS: P, S e Z
$CY = 0, A_c = 1$

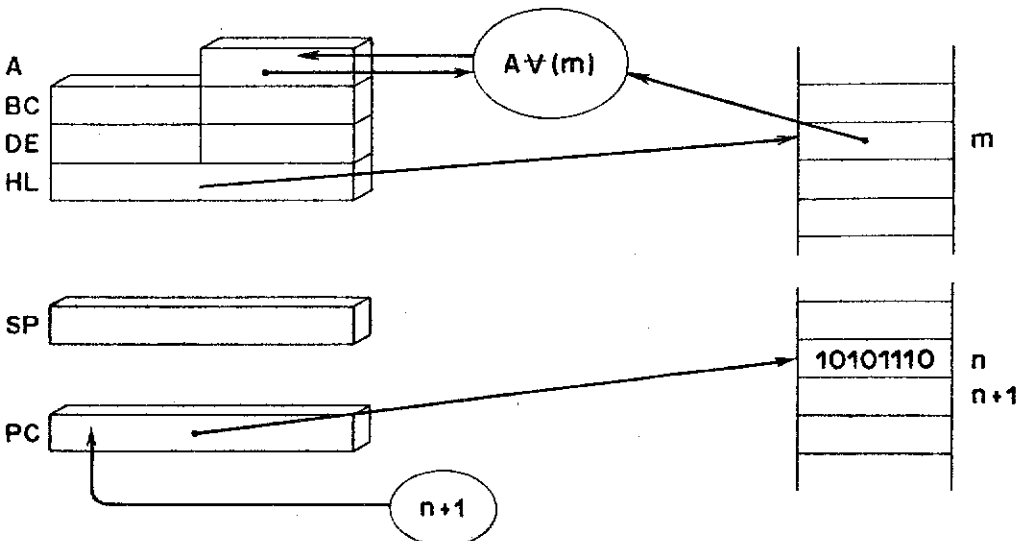
Ex.: Suponha $A = B9_{16}$
ANI 46H 11100110 01000110 o acumulador conterá 0
10111001 ∧ 01000110 → 00000000
$A_c = 1, CY = 0, P = 1, S = 0, Z = 1$

"OU EXCLUSIVO" COM REGISTRO — XRA r

Consiste em uma operação booleana *ou exclusivo* entre o conteúdo de um dos registros A, B, C, D, E, H ou L e o acumulador.

FLAGS: P, S e Z
$A_c = 0, CY = 0$

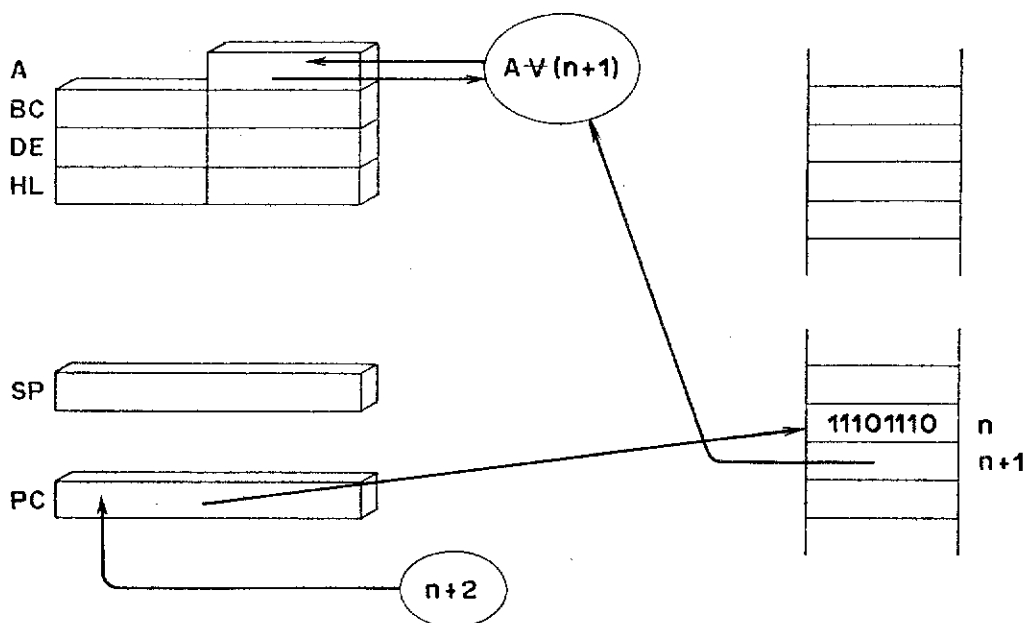
Ex.: Suponha A = $E3_{16}$ e C = $A0_{16}$
XRA C 10101001 o acumulador conterá 43_{16}
11100011 ∀ 10100000 → 01000011
$A_c = 0$, $CY = 0$, $P = 0$, $S = 0$, $Z = 0$

"OU EXCLUSIVO" COM POSIÇÃO DE MEMÓRIA — XRA M

Consiste em uma operação booleana *ou exclusivo* entre o conteúdo de uma posição de memória, cujo endereço se encontra no par de registro HL e o acumulador.

FLAGS: P, S e Z
$CY = 0, A_c = 0$

Ex.: Suponha $A = E3_{16}$, $HL = 3A53_{16}$ e $(3A53_{16}) = 5F_{16}$
XRA M 10100110 o acumulador conterá BC_{16}
11100011 $\forall$ 01011111 ⟶ 10111100
$A_c = 0, CY = 0, P = 0, S = 1, Z = 0$

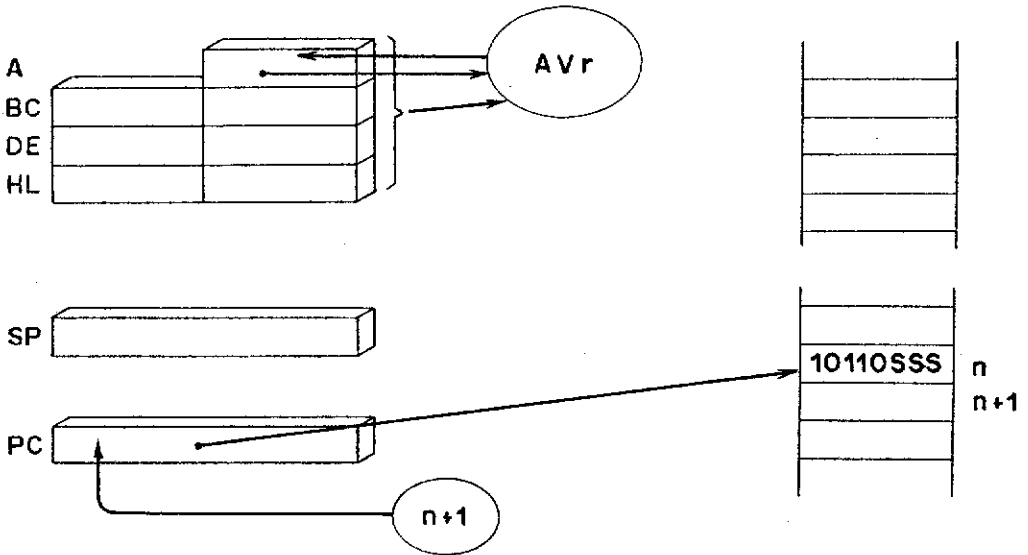
"OU EXCLUSIVO" IMEDIATO — XRI e

Consiste em uma operação booleana *ou exclusivo* entre o byte seguinte à instrução e o conteúdo do acumulador.

FLAGS: P, S e Z
$CY = 0, A_c = 0$

Ex.: Suponha $A = B9_{16}$
XRI 46H 11101110 01000110 o acumulador conterá FF_{16}
10111001 $\forall$ 01000110 ⟶ 11111111
$A_c = 0, CY = 0, P = 1, S = 1, Z = 0$

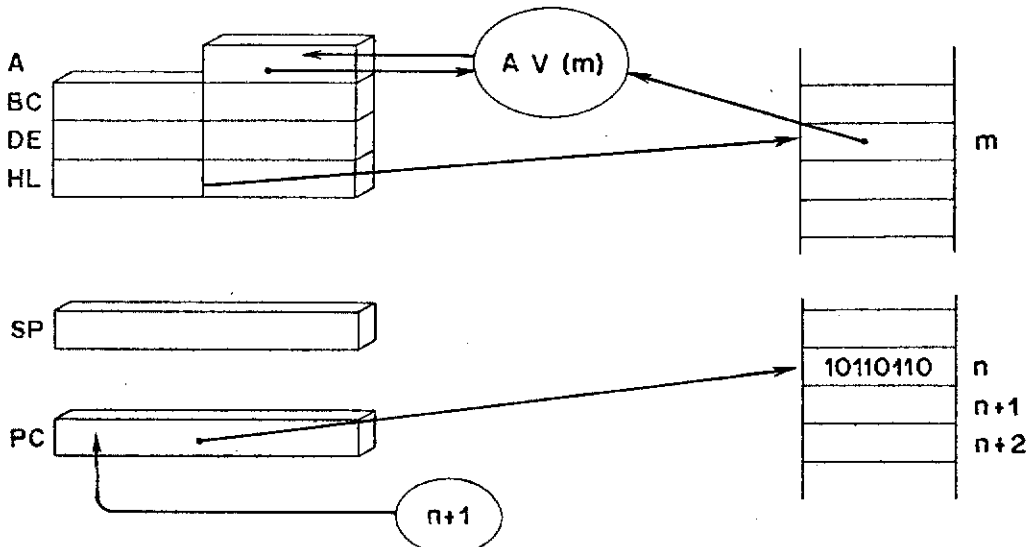
"OU" COM REGISTRO — ORA r

Consiste em uma operação booleana *ou* entre o conteúdo de um dos registros A, B, C, D, E, H ou L e o acumulador.

A
BC
DE
HL
A V r
SP
PC
10110SSS
n
n+1
n+1

FLAGS: P, S e Z
CY = 0, A_c = 0

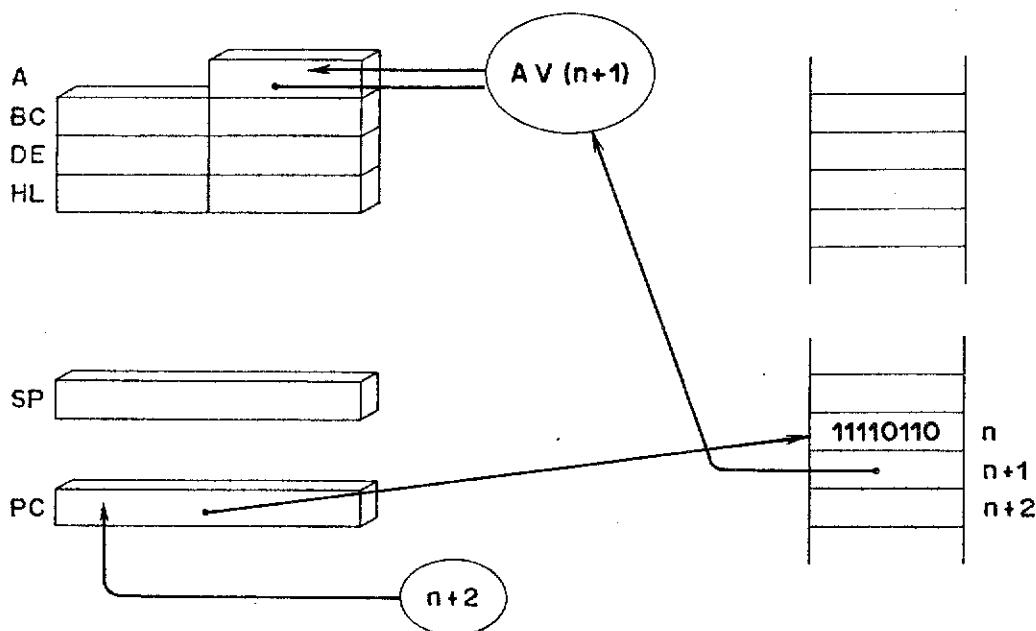
Ex.: Suponha A = $E3_{16}$ e C = $A0_{16}$
ORA C 10110001 o acumulador conterá $E3_{16}$
11100011 ∨ 10100000 ⟶ 11100011
A_c = 0, CY = 0, P = 0, S = 1, Z = 0

"OU" COM POSIÇÃO DE MEMÓRIA — ORA M

Consiste em uma operação booleana *ou* entre o conteúdo de uma posição de memória, cujo endereço se encontra no par de registros HL e o acumulador.

FLAGS: P, S e Z
CY = 0, A_c = 0

Ex.: Suponha A = $E3_{16}$, HL = $3A53_{16}$ e ($3A53_{16}$) = $5F_{16}$
ORA M 10110110 o acumulador conterá FF_{16}
11100011 ∨ 01011111 ⟶ 11111111
A_c = 0, CY = 0, P = 1, S = 1, Z = 0

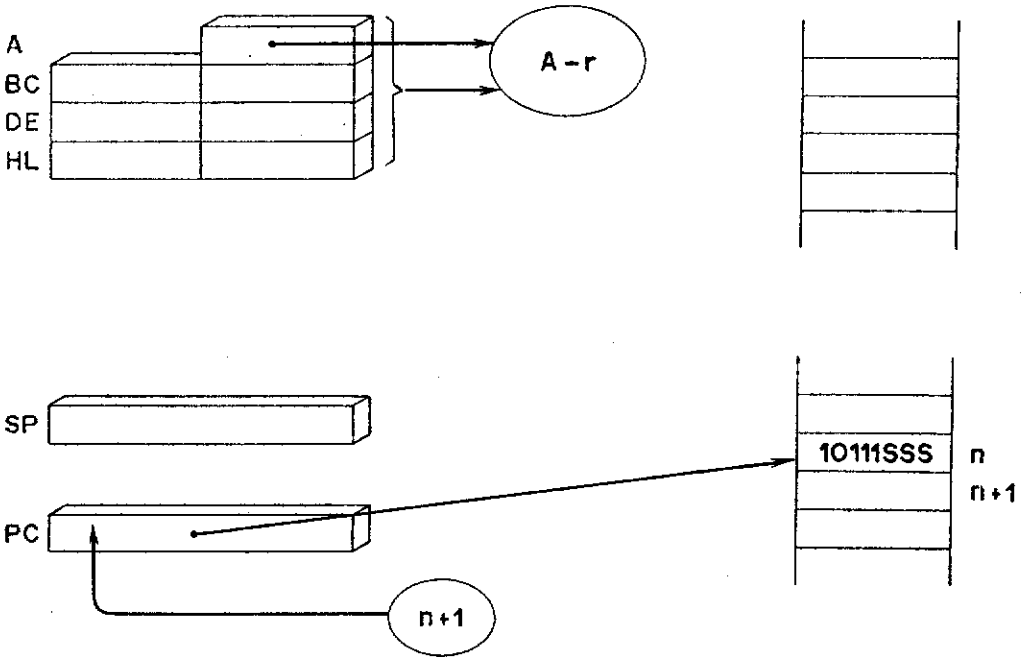
"OU" IMEDIATO — ORI e

Consiste em uma operação booleana *ou* entre o byte seguinte à instrução e o acumulador.

FLAGS: P, S e Z
CY = 0, A_c = 0

Ex.: Suponha A = $B9_{16}$
ORI 45H 11110110 01000101 o acumulador conterá FD_{16}
10111001 ∨ 01000101 ⟶ 11111101
A_c = 0, CY = 0, P = 0, S = 1, Z = 0

COMPARAÇÃO COM REGISTRO — CMP r

Consiste em subtrair o conteúdo de um dos registros: A, B, C, D, E, H ou L do acumulador. O conteúdo do acumulador não é alterado, e os flags passam

a refletir o resultado da subtração, isto é: se Z = 1, significa que A = r, se CY = 1, significa que A < r.

FLAGS: A_c, P, S, CY e Z

Ex.: Suponha que $A = F3_{16}$ e $B = B0_{16}$

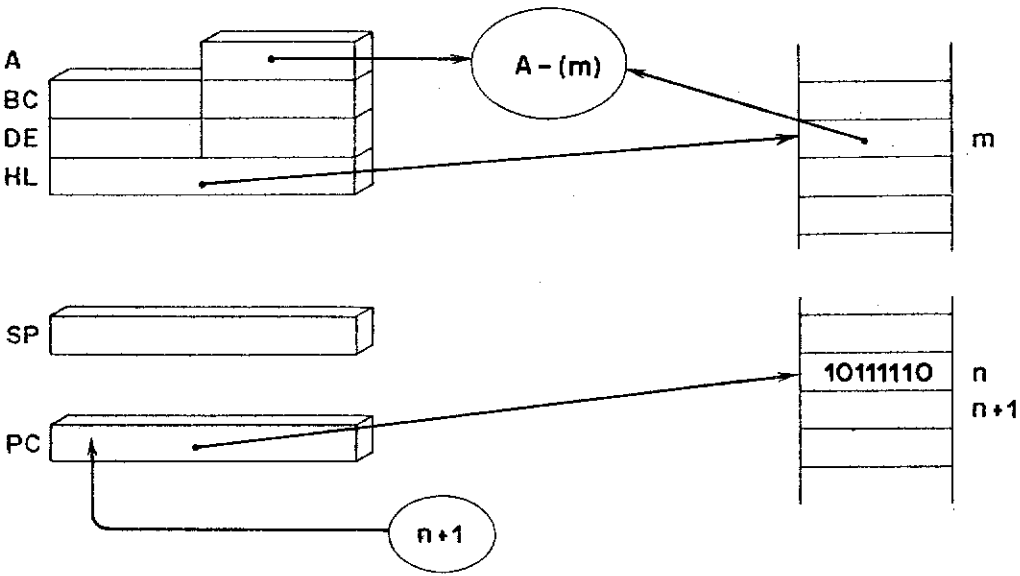
CMP B   10111000   o acumulador continuará com $F3_{16}$

$P = 0, Z = 0, A_c = 0, CY = 0, S = 0$

$F3_{16}$ ⟶ 11110011
Complemento a 2 de $B0_{16}$ ⟶ + 01010000
1 ⟵ 01000011

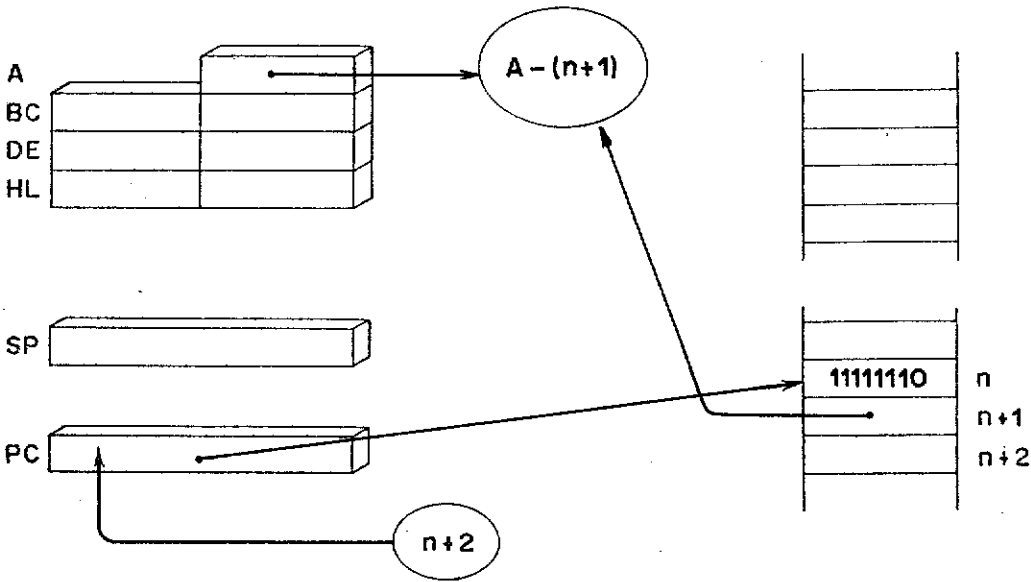
Observe que o flag VAI UM é complementado.

COMPARAÇÃO COM MEMÓRIA — CMP M

Consiste em subtrair o conteúdo de uma posição de memória, cujo endereço se encontra no par de registros HL, do acumulador. O acumulador não é alterado, e os flags passam a refletir o resultado da subtração, isto é: se Z = 1, significa A = (HL), se CY = 1, significa A < (HL).

FLAGS: A_c, P, S, CY e Z

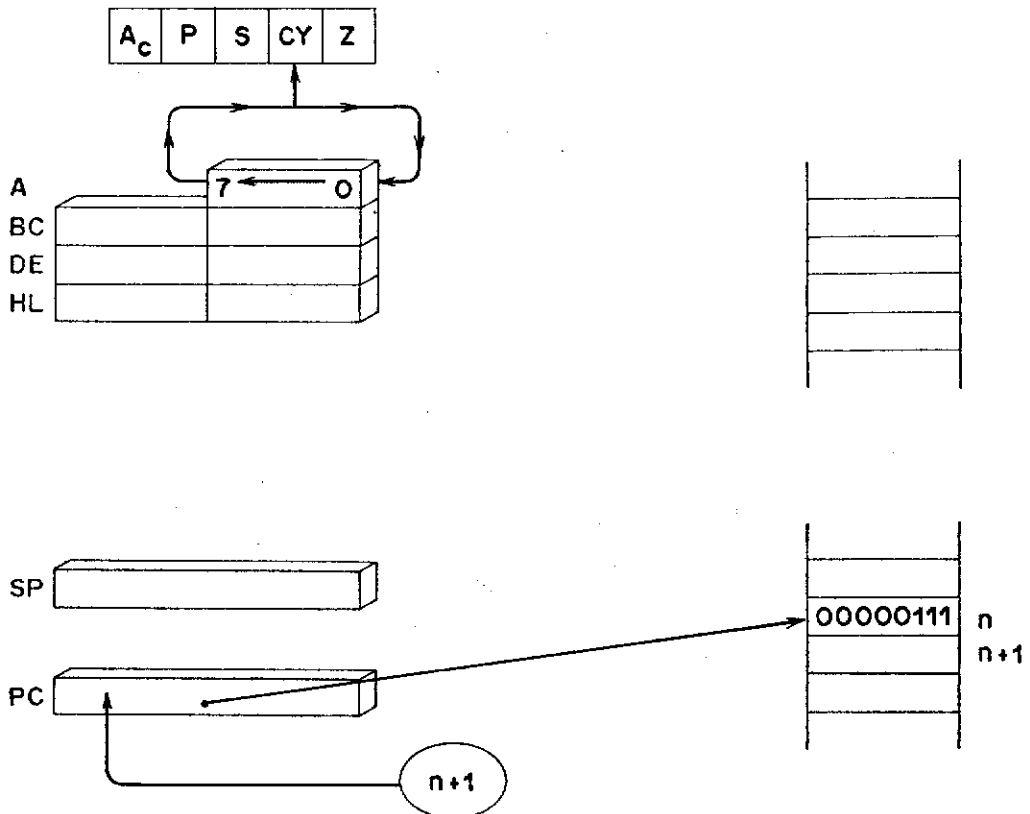
Ex.: Suponha: A = 09_{16}, HL = $01A7_{16}$ e $(01A7_{16})$ = $C8_{16}$
CMP M   10111110   o acumulador continuará 09_{16}
$A_c = 0$, $P = 1$, $S = 0$, $CY = 1$, $Z = 0$

COMPARAÇÃO IMEDIATA — CPI e

Consiste em subtrair o conteúdo do byte seguinte à instrução do acumulador. O acumulador não é alterado, e os flags passam a refletir o resultado da subtração.

FLAGS: A_c, P, S, CY e Z

Ex.: Suponha A = $7C_{16}$
CPI 3AH 11111110 00111010 o acumulador continuará em $7C_{16}$
$A_c = 0$, $P = 1$, $S = 0$, $CY = 0$, $Z = 0$

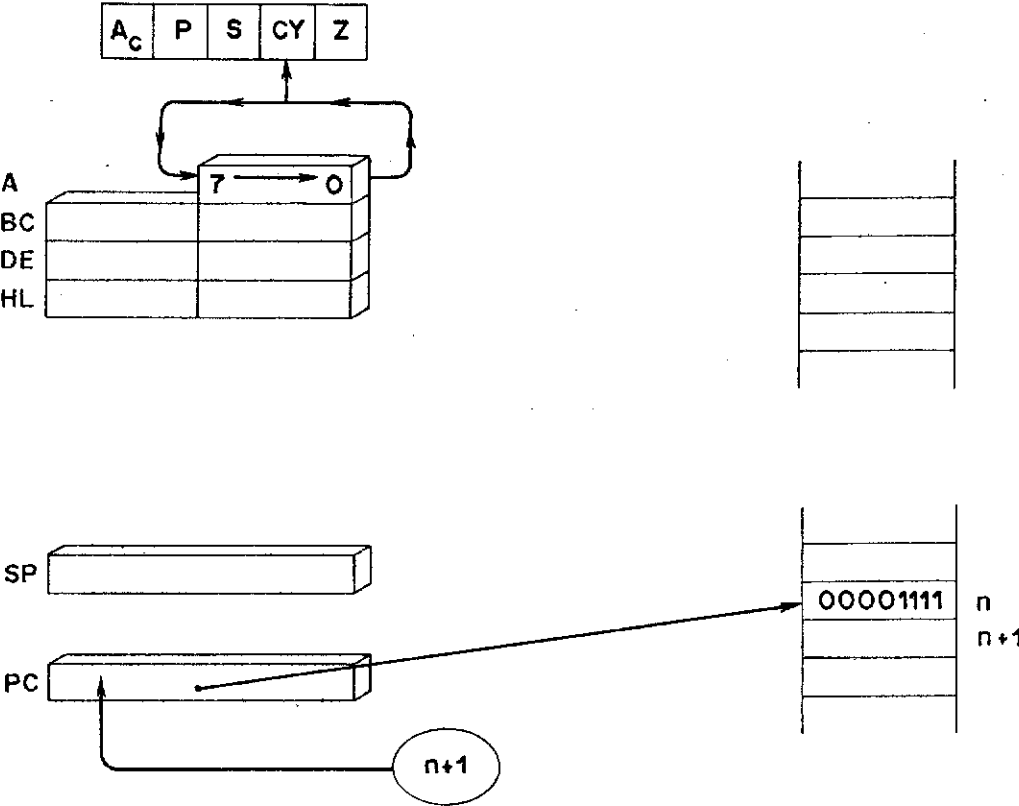
ROTAÇÃO À ESQUERDA — RLC

Consiste em deslocar o conteúdo do acumulador para a esquerda de um bit. O bit 0 do acumulador e o flag VAI UM recebem o bit de maior significado (bit 7).

FLAGS: CY

Ex.:

RLC	00000111	*ANTES*	*DEPOIS*
		A = 01010110	A = 10101100
		CY = 1	CY = 0

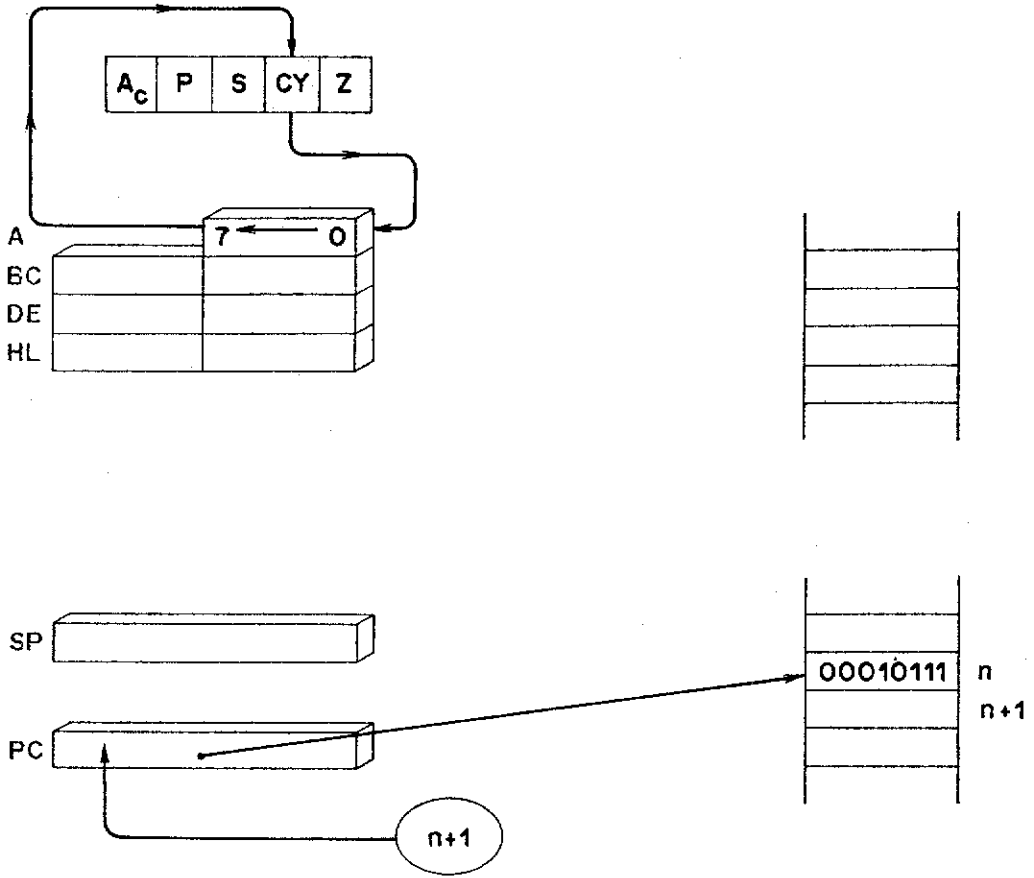
ROTAÇÃO À DIREITA — RRC

Consiste em deslocar o conteúdo do acumulador para a direita de um bit. O bit 7 do acumulador e o flag VAI UM recebem o bit de menor significado (bit 0).

FLAGS: CY

Ex.:

RRC	00001111	*ANTES*	*DEPOIS*
		A = 01010110	A = 00101011
		CY = 1	CY = 0

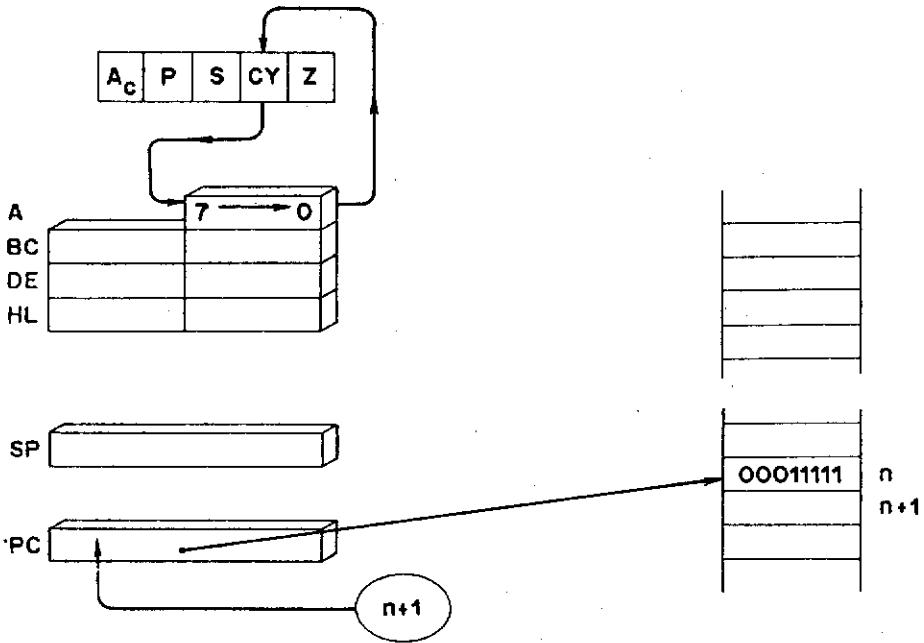
ROTAÇÃO À ESQUERDA ATRAVÉS DO FLAG VAI UM — RAL

Consiste em deslocar o conteúdo do acumulador para a esquerda de 1 bit através do flag VAI UM. O bit zero do acumulador recebe o VAI UM, e o VAI UM recebe o bit mais significativo (bit 7).

FLAGS: CY

Ex.:

RAL	00010111	*ANTES*	*DEPOIS*
		A = 01010110	A = 10101101
		CY = 1	CY = 0

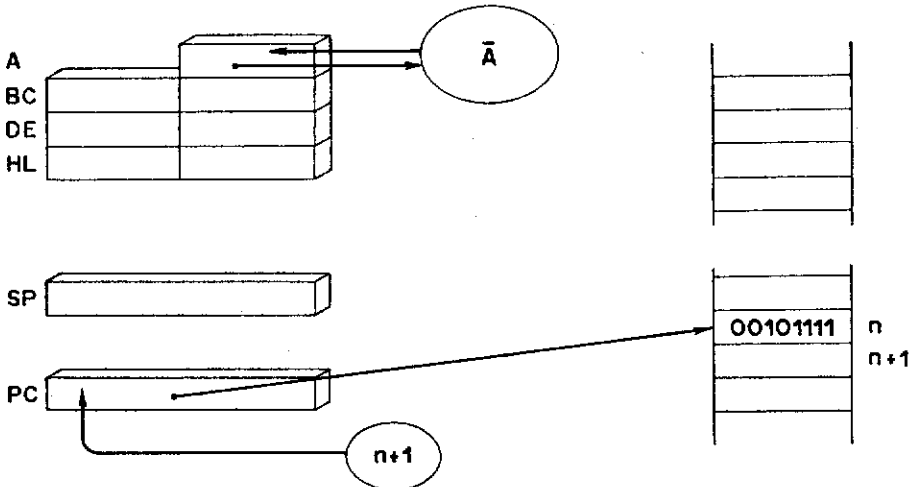
ROTAÇÃO À DIREITA ATRAVÉS DO FLAG VAI UM — RAR

Consiste em deslocar o conteúdo do acumulador para a direita de 1 bit através do flag VAI UM. O bit 7 recebe o conteúdo do VAI UM e o VAI UM recebe o bit menos significativo (bit 0).

FLAGS: CY

Ex.:

RAR 00011111	*ANTES*	*DEPOIS*
	A = 01010110	A = 10101011
	CY = 1	CY = 0

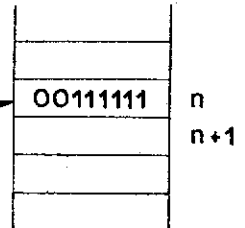
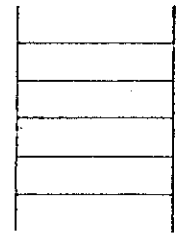
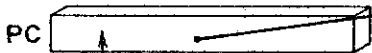
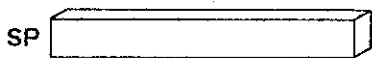
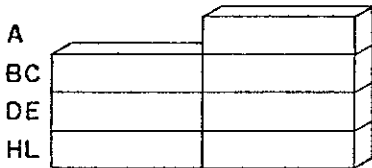
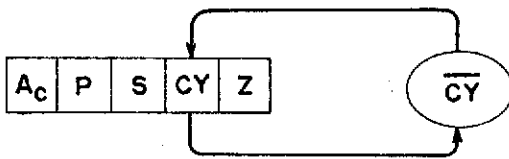
COMPLEMENTO DO ACUMULADOR — CMA

Consiste em complementar o conteúdo do acumulador: bits zero são trocados por um, e bits um são trocados por zero.

FLAGS: ______

Ex.: Suponha A = $B6_{16}$
CMA   00101111   o acumulador conterá 49_{16}

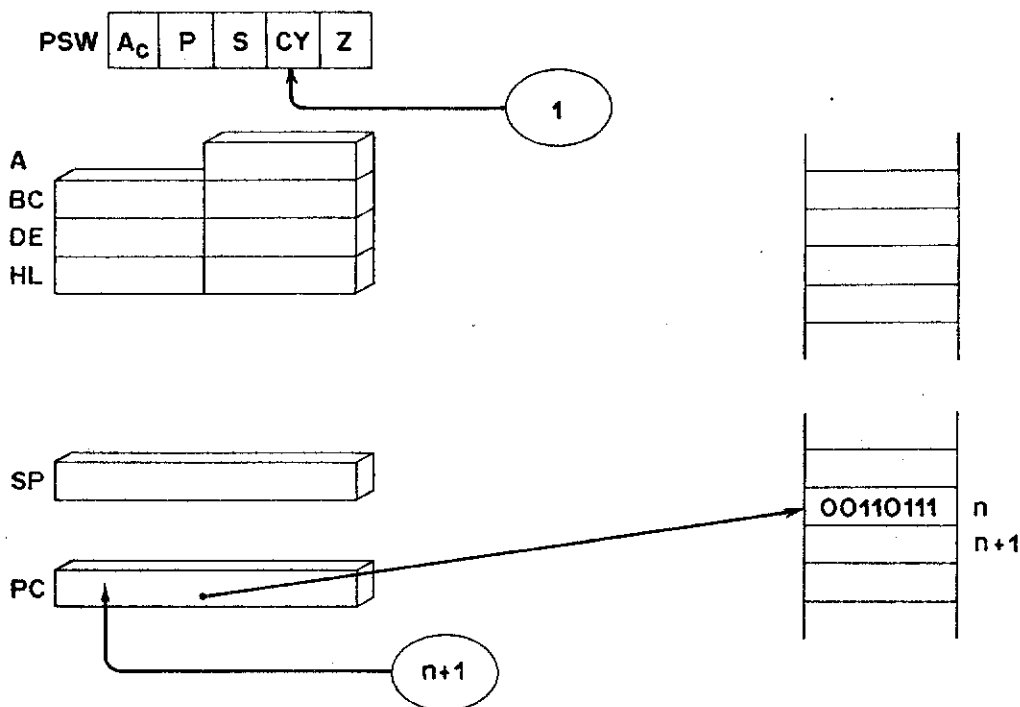
acumulador { antes → 10110110
             depois → 01001001

COMPLEMENTO DO FLAG VAI UM — CMC

Consiste em complementar o flag VAI UM. Se o VAI UM é um, passa a zero; se é zero, passa a um.

FLAGS: CY

Ex.: Suponha que CY = 0
CMC   00111111   o CY será igual a 1

LIGA FLAG VAI UM — STC

Consiste em forçar o flag VAI UM para condição de ligado, isto é, CY recebe o valor 1.

FLAGS: CY

Ex.:

STC 00110111 independente do conteúdo anterior de CY, após a instrução o conteúdo passa a ser 1
utilizada em conjunto com instrução CMC, é possível forçar uma condição de desligado para o flag VAI UM.

STC 00110111
CMC 00111111

4. INSTRUÇÕES DE DESVIO

Instruções de desvio são aquelas que alteram o fluxo seqüencial do programa. Temos dois tipos: incondicional e condicional.

Incondicional — simplesmente altera o PC (Apontador de Programa)
Condicional — examina o "status" de um dos quatro flags (P, S, CY e Z), para determinar se o desvio será executado ou não.

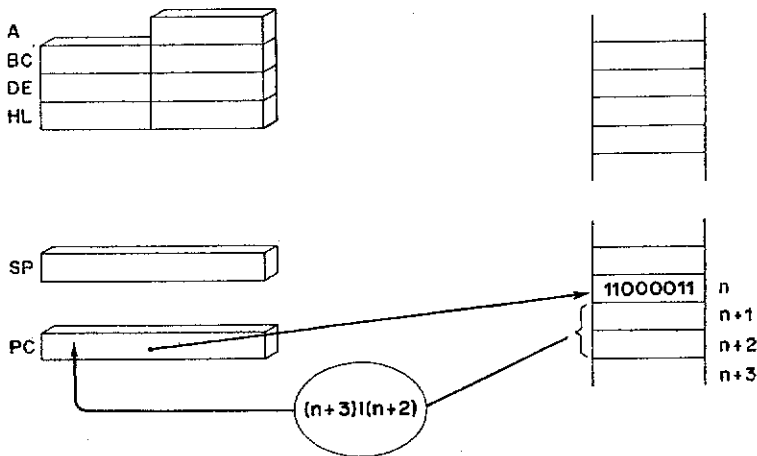
As condições testadas são as seguintes:

NZ — não zero (Z = 0)
Z — zero (Z = 1)
NC — não VAI UM (CY = 0)
C — VAI UM (CY = 1)

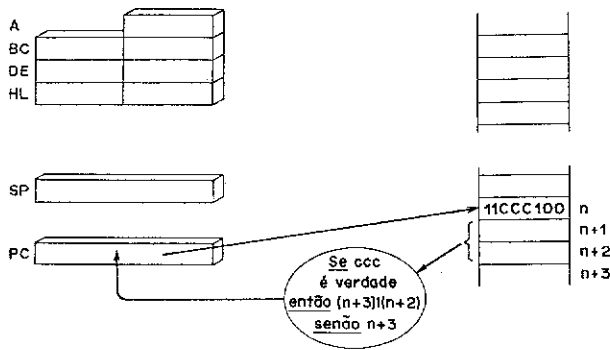
PO — paridade ímpar (P = 0)
PE — paridade par (P = 1)
P — positivo (S = 0)
M — negativo (S = 1)

Os flags não são alterados por nenhuma instrução de desvio.

DESVIO INCONDICIONAL — JMP end

O Apontador de Programa (PC) é alterado para o valor indicado nos dois bytes seguintes à instrução. A parte baixa do novo PC está no segundo byte da instrução e a parte alta no terceiro.

FLAGS: —

Ex.: Suponha PC = 100_{16}
JMP 02F2H 11000011 00000010 o PC passará a apontar a posição de memória $2F2_{16}$

Consiste em alterar o conteúdo do PC para o endereço especificado nos dois bytes seguintes à instrução quando a condição especificada for verdadeira, em caso contrário o fluxo normal do programa não é alterado.

FLAGS: —

JNZ	11 000 010	desvia somente quando Z = 0
JZ	11 001 010	desvia somente quando Z = 1
JNC	11 010 010	desvia somente quando CY = 0
JC	11 011 010	desvia somente quando CY = 1
JPO	11 100 010	desvia somente quando P = 0
JPE	11 101 010	desvia somente quando P = 1
JP	11 110 010	desvia somente quando S = 0
JM	11 111 010	desvia somente quando S = 1

Ex.:

100_{16}	JNC	120H	11010010 00100000 00000001
103_{16}	MOV	A,C	
.	—		
.	—		
.	—		
120_{16}	ANI	FFH	

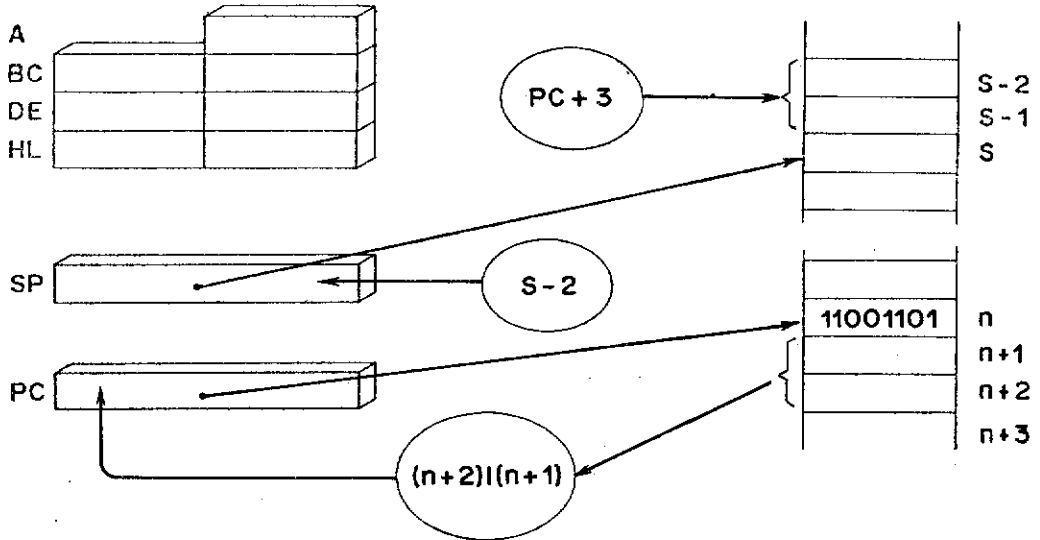
Se CY = 0 *então* PC ← 120_{16} (executa ANI FFH)
senão PC ← 103_{16} (executa MOV A,C)

10_{16}	MVI	B,10
11_{16}	MOV	A,M
.	—	
.	—	
.	—	
$1F_{16}$	DCR	B
20_{16}	JNZ	11H
.	—	
.	—	
.	—	

Se Z = 0 *então* PC ← 11_{16} (executa MOV A,M)
senão PC ← 23_{16}

CHAMADA INCONDICIONAL DE SUBPROGRAMAS — CALL end

Consiste em desviar o controle de execução para um trecho de programa denominado subprograma, sendo que o endereço de continuação do fluxo normal é guardado na pilha.

A parte alta do endereço que se encontra no PC é guardada em uma posição de memória apontada por SP-1, e a parte baixa apontada por SP-2. O conteúdo de SP é decrementado de 2, e o controle é transferido para instrução cujo endereço está especificado nos dois bytes seguintes a instrução.

FLAGS: —.

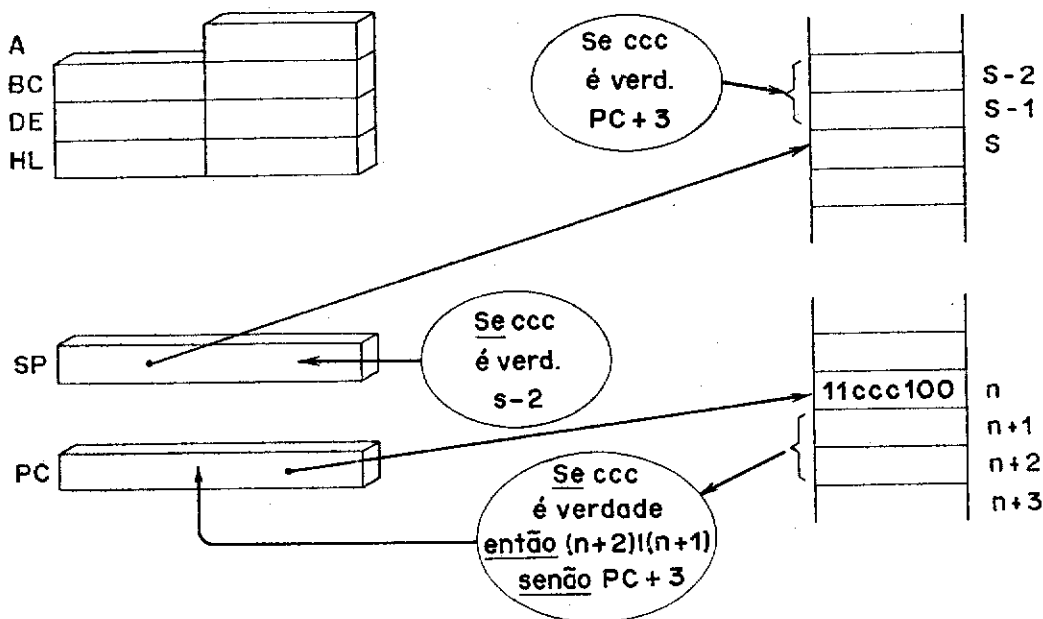
Ex.:

. —
. —
. —

115_{16} CALL SUB 11001101 10100000 00000010
118_{16} MOV A,0
.
.
.
$2A0_{16}$ SUB —
. —
. —
. —

ANTES		*DEPOIS*	
PC	= 115_{16}	PC	= 240_{16}
SP	= 700_{16}	SP	= $6FE_{16}$
($6FF_{16}$)	= ?	($6FF_{16}$)	= 1
($6FE_{16}$)	= ?	($6FE_{16}$)	= 18_{16}

Consiste em desviar o controle de execução para um trecho de programa denominado subprograma, caso a condição indicada seja verdadeira.

A parte alta do endereço que se encontra no PC é guardada em uma posição de memória apontada por SP-1, e a parte baixa em SP-2. O conteúdo de SP é decrementado de 2, e o controle é transferido para instrução cujo endereço está especificado nos dois bytes seguintes à instrução.

FLAGS: ——

CNZ	11 000 100	desvia para o subprograma somente quando Z = 0
CZ	11 001 100	desvia para o subprograma somente quando Z = 1
CNC	11 010 100	desvia para o subprograma somente quando CY = 0
CC	11 011 100	desvia para o subprograma somente quando CY = 1
CPO	11 100 100	desvia para o subprograma somente quando P = 0
CPE	11 101 100	desvia para o subprograma somente quando P = 1
CP	11 110 100	desvia para o subprograma somente quando S = 0
CM	11 111 100	desvia para o subprograma somente quando S = 1

Ex.:

```
.                  —
.                  —
.                  —
115₁₆              CNC ROT    11010100 10100000 00000010
118₁₆              MOV A0
.                  —
.                  —
.                  —
2A0      ROT:      —
```

ANTES

PC $= 115_{16}$
SP $= 700_{16}$
$(6FF_{16}) = ?$
$(6FE_{16}) = ?$

DEPOIS

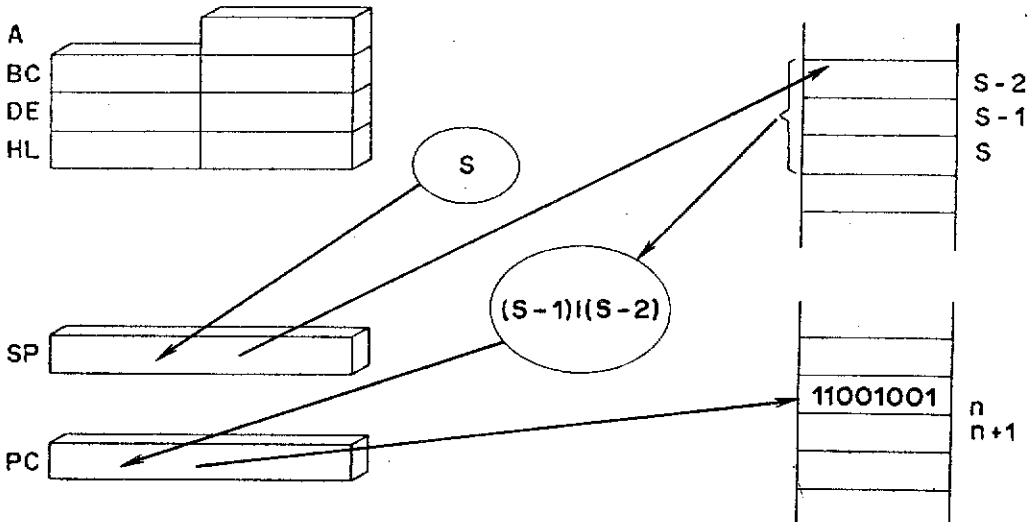
Se CY = 0 *então* { PC $= 2A0_{16}$; SP $= 6FE_{16}$; $(6FF_{16}) = 1$; $(6FE_{16}) = 18_{16}$ }

senão { PC $= 118_{16}$; SP $= 700_{16}$; $(6FF_{16}) = ?$; $(6FE_{16}) = ?$ }

RETORNO DE SUBPROGRAMA — RET

Consiste em retornar o controle de execução de um subprograma para o fluxo normal.

O conteúdo dos dois bytes que se encontram no topo da PILHA correspondem ao novo PC. O Apontador de Pilha (SP) é incrementado de 2.

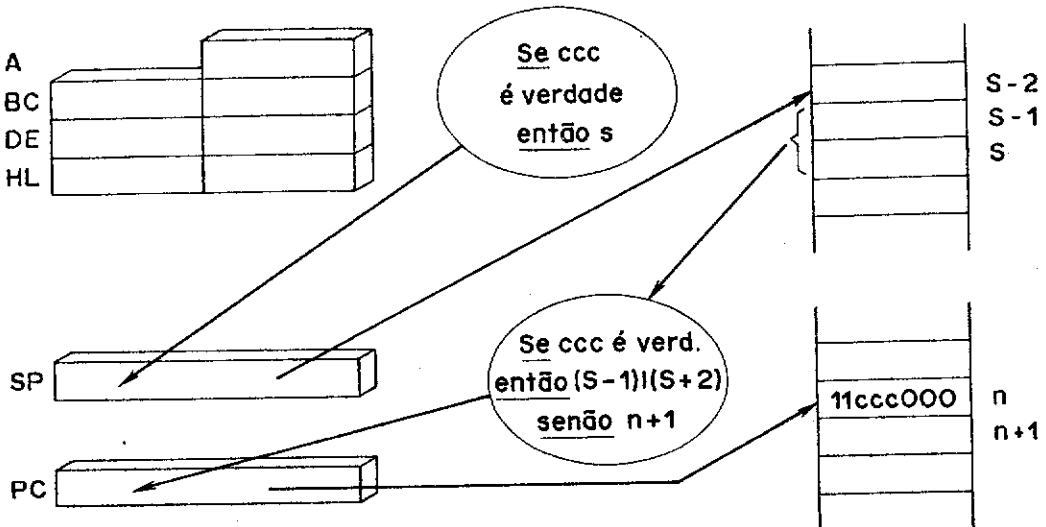
Todo subprograma deve terminar por uma instrução RET ou equivalente para que o controle de execução retorne ao programa de origem.

FLAGS: —

Ex.:
RET 11001001

ANTES		*DEPOIS*
PC	$= 2AF_{16}$	PC $= 118_{16}$
SP	$= 6FE_{16}$	SP $= 700_{16}$
$(6FF_{16})$	$= 1$	
$(6FE_{16})$	$= 18_{16}$	

RETORNO CONDICIONAL DE SUBPROGRAMAS —	RNZ RZ RNC RC RPO RPE RP RM

Consiste em retornar o controle de execução de um trecho de programa denominado subprograma, para o fluxo normal caso a condição específica seja verdadeira.

O conteúdo dos dois bytes que se encontram no topo da PILHA corresponderá ao novo PC. O Apontador de Pilha (SP) é incrementado de 2.

FLAGS: —

RNZ	11[000]000	retorna do subprograma somente quando Z = 0
RZ	11[001]000	retorna do subprograma somente quando Z = 1
RNC	11[010]000	retorna do subprograma somente quando CY = 0

RC	11[011]000	retorna do subprograma somente quando CY = 1
RPO	11[100]000	retorna do subprograma somente quando P = 0
RPE	11[101]000	retorna do subprograma somente quando P = 1
RP	11[110]000	retorna do subprograma somente quando S = 0
RM	11[111]000	retorna do subprograma somente quando S = 1

Ex.:
RM 11111000

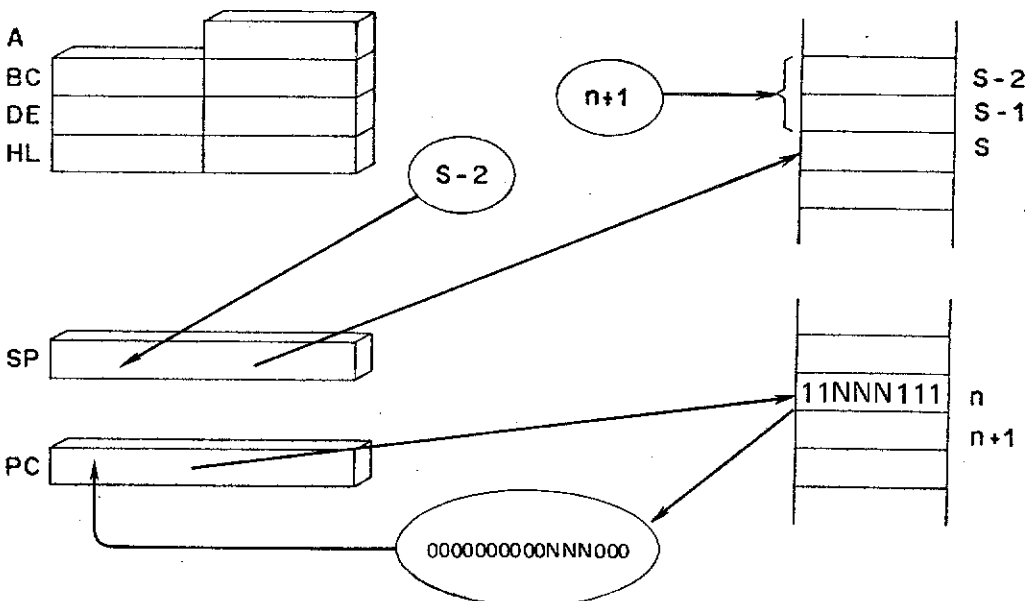
ANTES
PC $= 2AF_{16}$
SP $= 6FE_{16}$
$(6FF_{16}) = 1$
$(6FE_{16}) = 18_{16}$

DEPOIS
Se S = 1 *então* $\begin{cases} PC = 118_{16} \\ SP = 700_{16} \end{cases}$

senão $\begin{cases} PC = 2B0_{16} \\ SP = 6FE_{16} \\ (6FF_{16}) = 1_{16} \\ (6FE_{16}) = 18_{16} \end{cases}$

RESTART — RST n

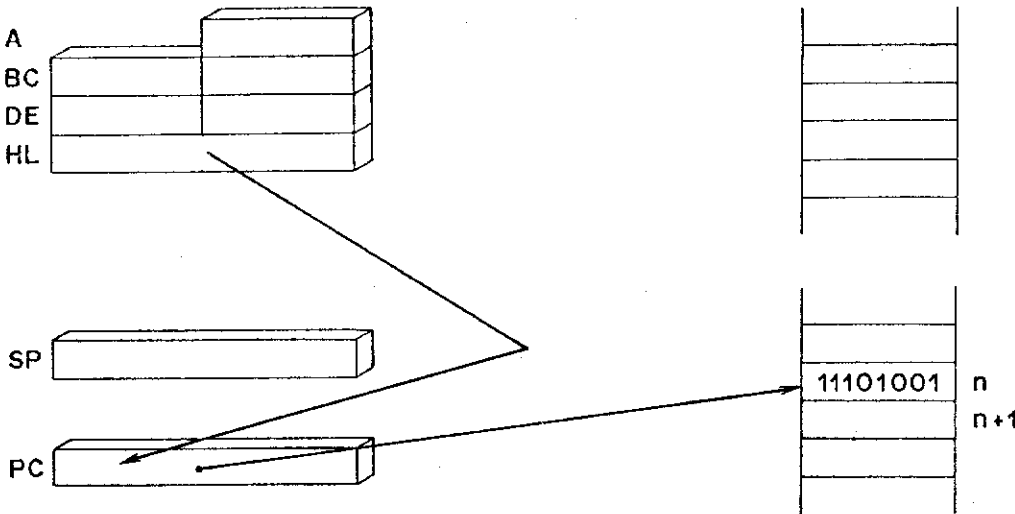
Consiste em uma chamada de subprograma especial de um byte, utilizada nos esquemas de interrupção.

O endereço da próxima instrução a ser executada é guardado na pilha sendo a parte alta em SP-1 e a parte baixa em SP-2. SP é decrementado de 2, e o controle (PC) é transferido para um dos seguintes endereços prévios: 0, 8, 16, 24, 32, 40, 48 ou 56:

FLAGS: ———

	NNN	*end* (base 16)
RST 0	000	00
RST 1	001	08
RST 2	010	10
RST 3	011	18
RST 4	100	20
RST 5	101	28
RST 7	110	30
RST 6	111	38

Ex.:
RST 2 11010111
É chamado o subprograma que se encontra na posição de memória 10_{16}.

DESVIO PARA CONTEÚDO DE HL — PCHL

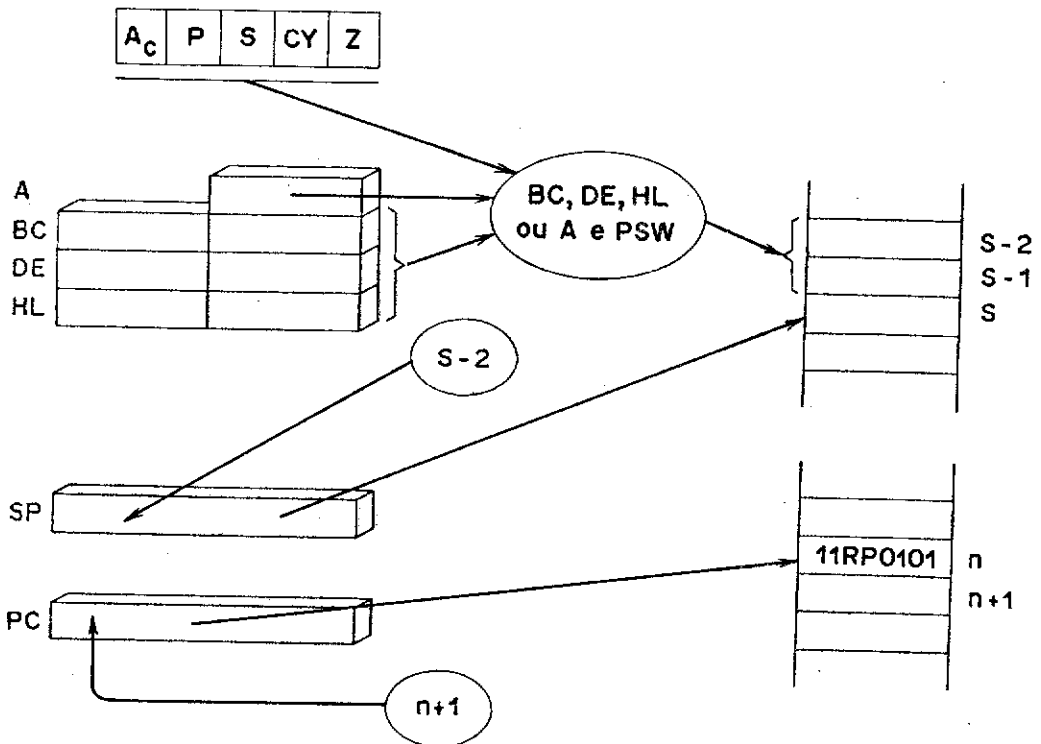
Consiste em desviar para o endereço que se encontra no par HL. O conteúdo do registro H passa a ser a parte alta do PC, e o conteúdo do registro L passa a ser a parte baixa.

FLAGS: ———

Ex.:
LXI H,2A0H
PCHL 11101001 a próxima instrução a ser executada encontra-se na posição de memória $2A0_{16}$.

Observe que o trecho acima é equivalente a JMP 2A0H.

5. INSTRUÇÕES DE MANIPULAÇÃO COM A PILHA, E/S, INTERRUPÇÕES E OUTRAS

ESCREVER NO TOPO DA PILHA — PUSH rp PUSH PSW

Consiste em salvar na pilha um dos pares de registros BC, DE, HL ou o acumulador e a PSW. O conteúdo de B, D, H ou A é movido para posição de memória cujo endereço é dado por SP-1. O conteúdo de C, E, L ou PSW é movido para posição de memória cujo endereço é dado por SP-2. O conteúdo do apontador da PILHA (SP) é decrementado de 2.

FLAGS: ______

Ex.:
PUSH PSW 11110101

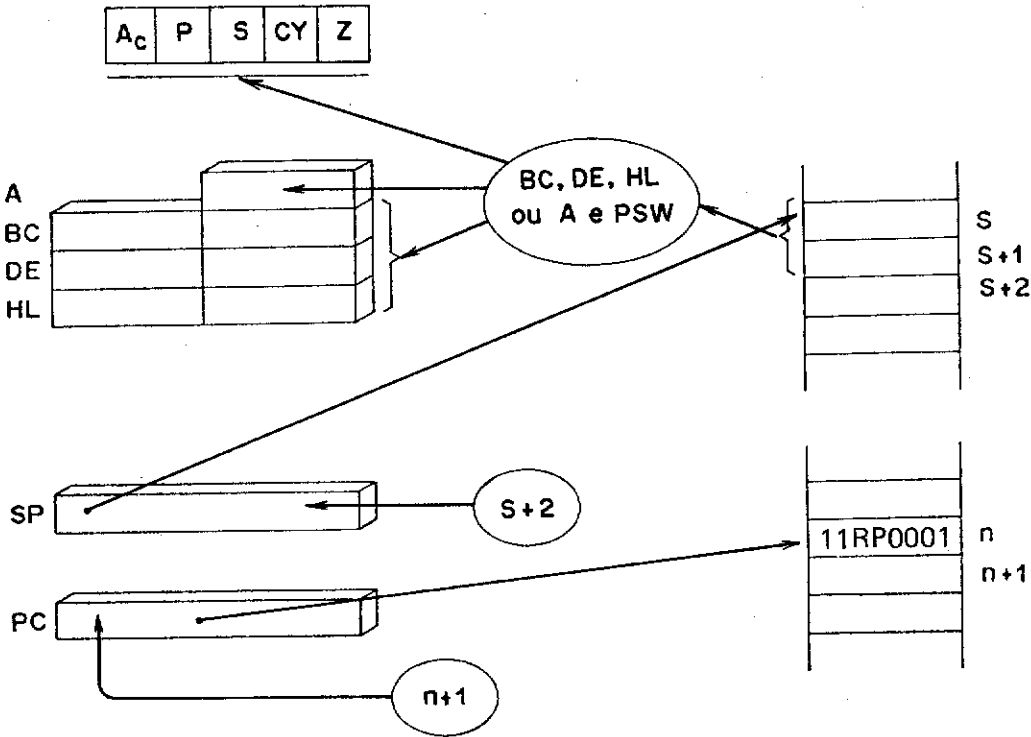
ANTES

A = FF_{16}
SP = 700_{16}
PSW = 10×0×1×1
x = indefinido

DEPOIS

A = FF_{16}
SP = $6FE_{16}$
($6FF_{16}$) = FF_{16}
($6FE_{16}$) = 10×0×1×1
x = indefinido

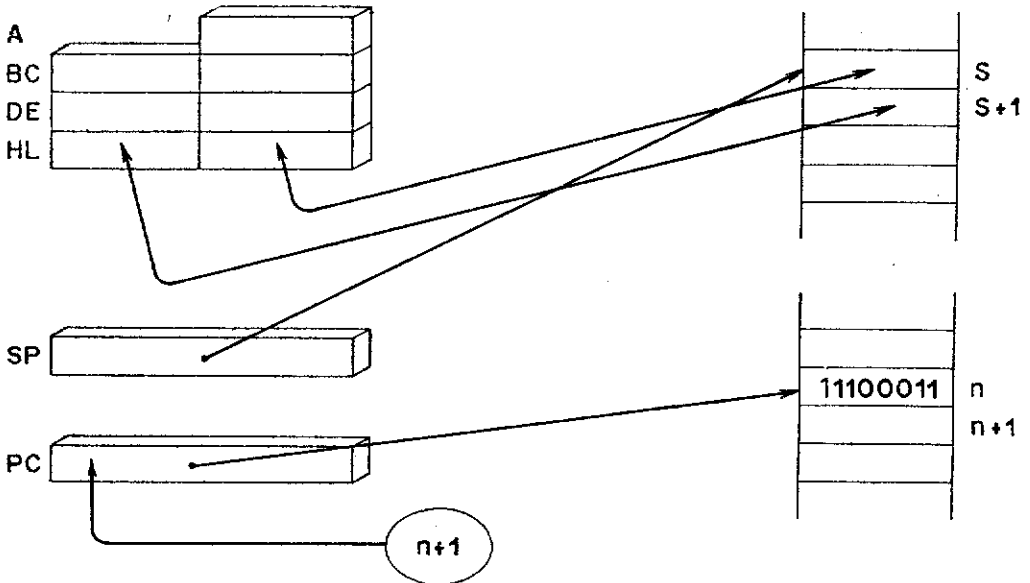
LER NO TOPO DA PILHA — POP rp POP PSW

Consiste em recuperar da PILHA um dos pares de registros BC, DE, HL, ou o acumulador e a PSW. O conteúdo da posição de memória referente ao topo da PILHA é movido para um dos registros C, E, L ou os flags de condição são alterados, a posição subseqüente é utilizada para alterar B, D, H ou A.

FLAGS: Somente são alterados com POP PSW

Ex.:
POP B 11000001

ANTES		*DEPOIS*
BC	= ?	BC = $70A3_{16}$
SP	= $6FE_{16}$	SP = 700_{16}
$(6FE_{16})$	= $A3_{16}$	
$(6FF_{16})$	= 70_{16}	

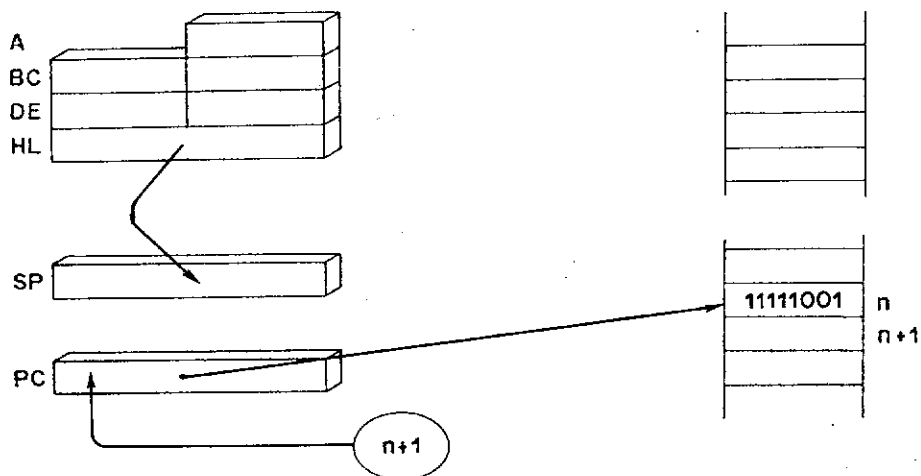
TROCA TOPO DA PILHA COM HL — XTHL

Consiste em trocar o conteúdo do par HL com o topo da PILHA e o byte subseqüente. O conteúdo do registro L é trocado com o conteúdo da posição de memória cujo endereço é especificado pelo conteúdo do registro SP, o conteúdo do registro H é trocado com a posição subseqüente de memória.

FLAGS: ______

Ex.:
XTHL 11100011

ANTES		*DEPOIS*	
HL	= $3A0_{16}$	HL	= $4B1_{16}$
SP	= 700_{16}	SP	= 700_{16}
(700_{16})	= $B1_{16}$	(700_{16})	= $A0_{16}$
$(6FF_{16})$	= 4	$(6FF_{16})$	= 3

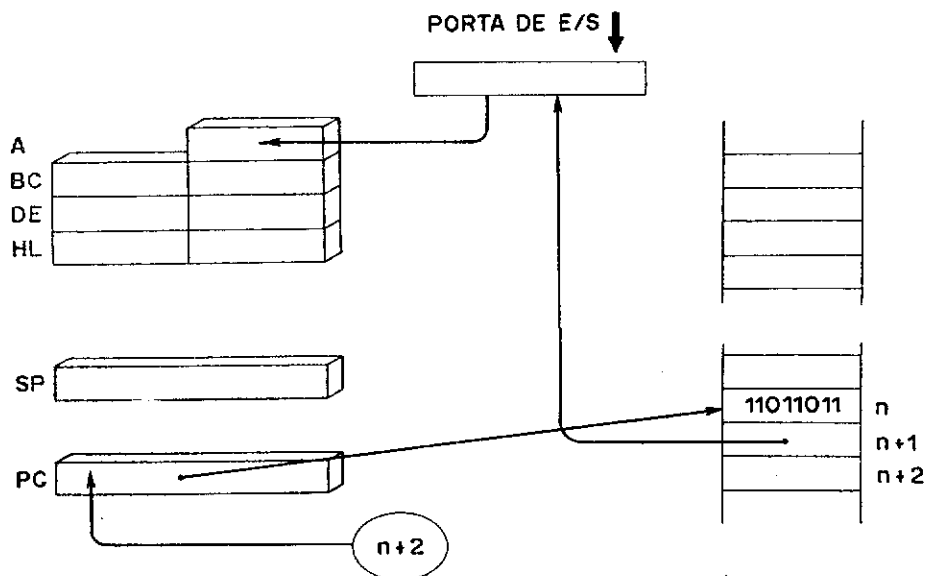
CARGA DE SP COM CONTEÚDO DE HL — SPHL

Consiste em definir o Apontador da Pilha (SP) com o conteúdo do par de registros HL.

FLAGS: ______

Ex.:
SPHL 11111001

ANTES	*DEPOIS*
H = 07	H = 7
L = 00	L = 0
SP = ?	SP = 700_{16}

ENTRADA — IN porta E/S

Consiste em mover para o acumulador o conteúdo de uma porta de E/S, identificada no segundo byte da instrução.

FLAGS: ______

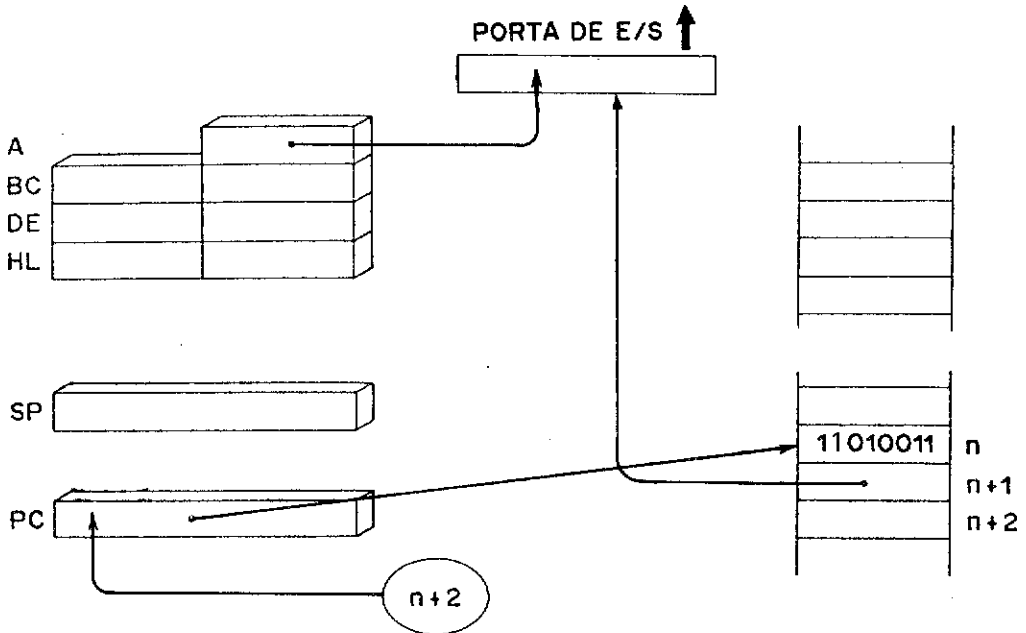
Ex.:
IN 1AH 11011011 00011010

ANTES
Porta E/S $1A_{16}$ = $F3_{16}$
A = ?

DEPOIS
A = $F3_{16}$

A instrução IN depende do microcomputador utilizado, e as portas lógicas são definidas na fase de projeto do *hardware*.

SAÍDA — OUT porta de E/S

Consiste em mover para uma porta de E/S, identificada no segundo byte da instrução, o conteúdo do acumulador.

Ex.:
OUT 1AH 11010011 00011010

ANTES
Porta de E/S $1A_{16}$ = ?
A = $F3_{16}$

DEPOIS
Porta de E/S — $F3_{16}$

A instrução OUT depende do microcomputador utilizado, e as portas lógicas são definidas na fase do projeto do *hardware*.

***HALT* — HLT**

Consiste em parar o processamento para reiniciar o processo; é necessário uma interrupção ou RESET.

A
BC
DE
HL

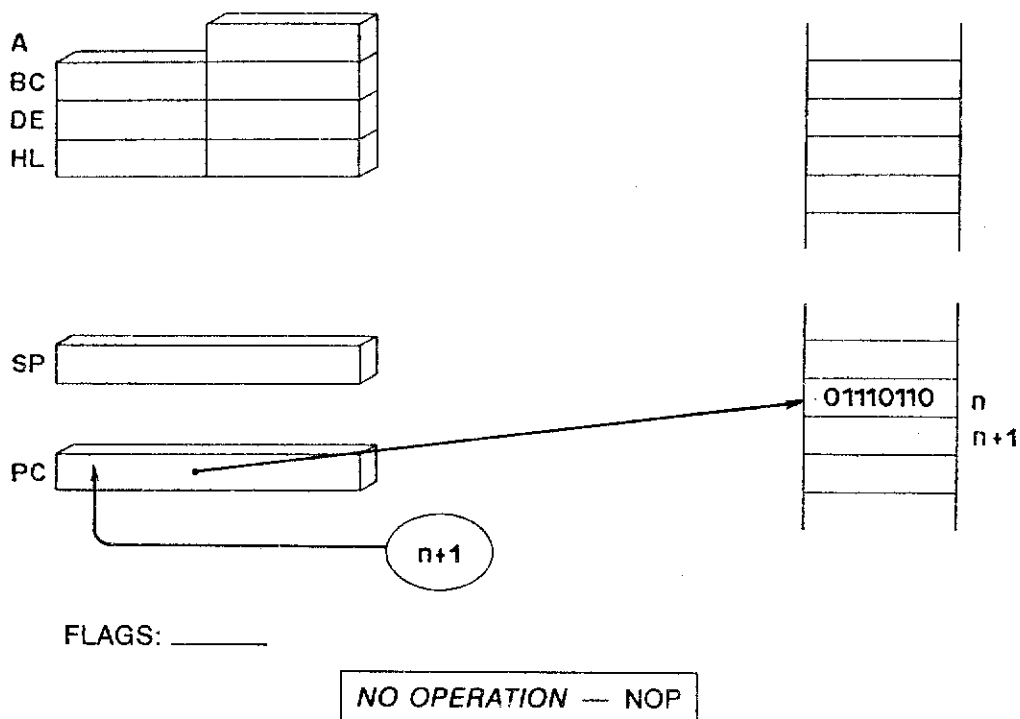
SP

01110110 n
n+1

PC

n+1

FLAGS: ______

NO OPERATION — NOP

Nada acontece quando esta instrução é executada.

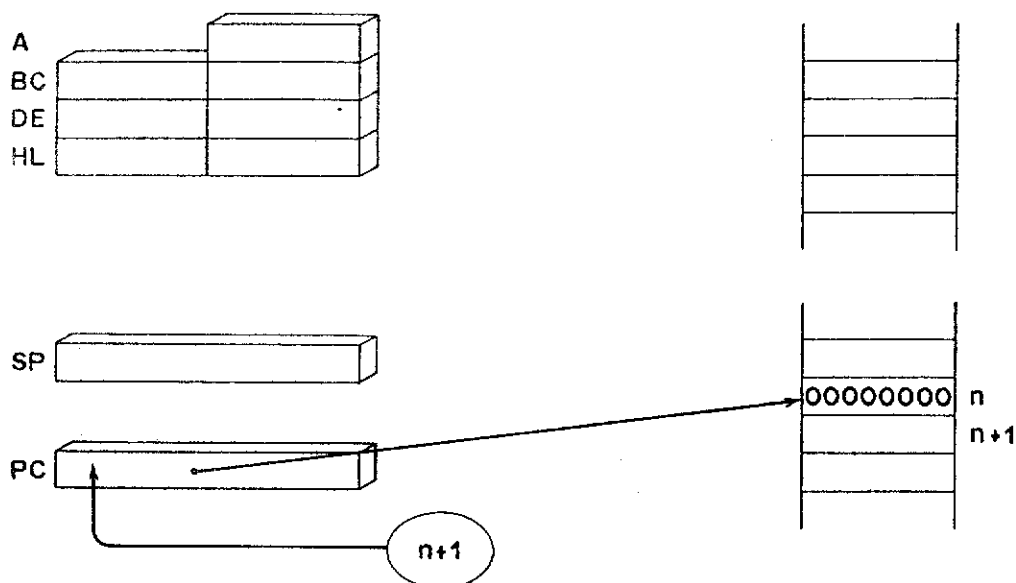
Esta instrução não é muito útil, porém pode ser utilizada para suprimir trechos que não devam ser executados, em caso de depuração direta do código objeto, além de poder ser utilizada para "loops" de espera (cada instrução consome 4 ciclos de clock).

A
BC
DE
HL

SP

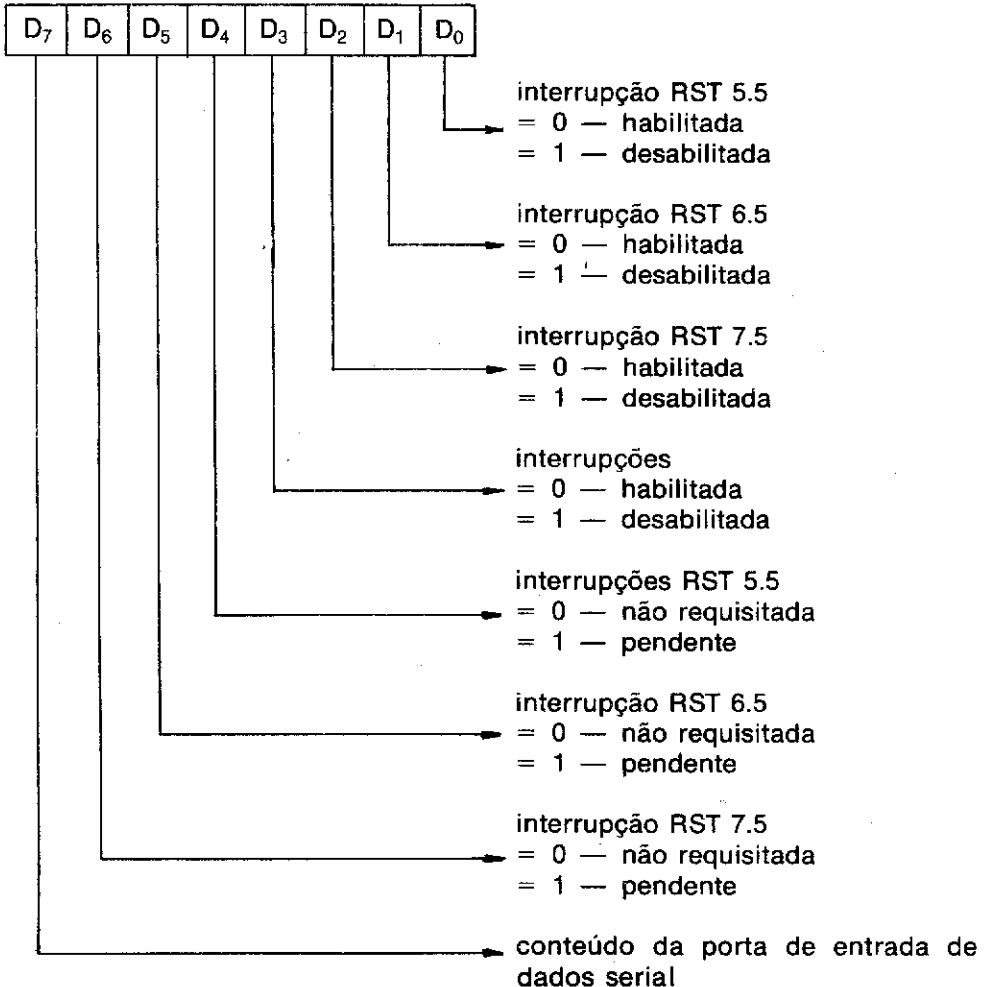
00000000 n
n+1

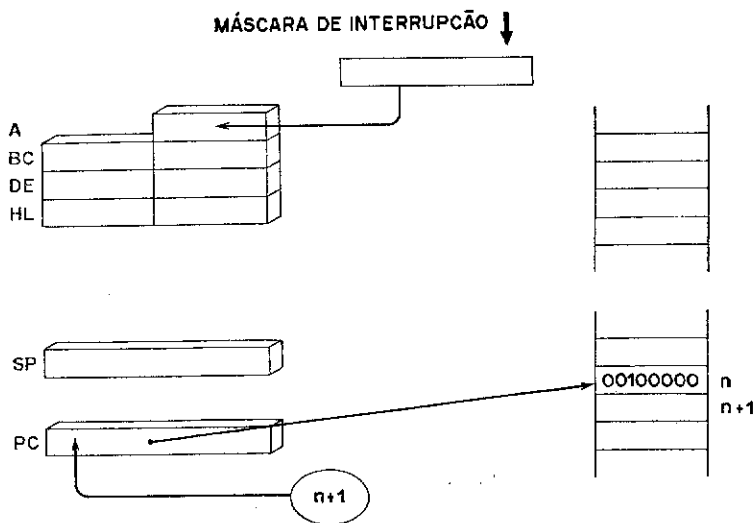
PC

n+1

FLAGS: ______

Ex.:
NOP 00000000

LÊ MÁSCARA DE INTERRUPÇÕES — RIM

Lê a máscara de interrupção. O acumulador recebe uma máscara de interrupção que obedece ao seguinte formato:

As interrupções RST 5.5, RST 6.5 e RST 7.5 funcionam de maneira similar as instruções RST 0 a RST 7, porém o endereço de desvio é o seguinte:

RST 5.5 — $002C_{16}$
RST 6.5 — 0034_{16}
RST 7.5 — $003C_{16}$

ESCREVE MÁSCARA DE INTERRUPÇÕES — SIM

Define a máscara de interrupção. O conteúdo do acumulador passa a definir a máscara de interrupção que passará a ser utilizada, e envia um sinal para a porta de saída de dados serial. O formato de máscara é o seguinte:

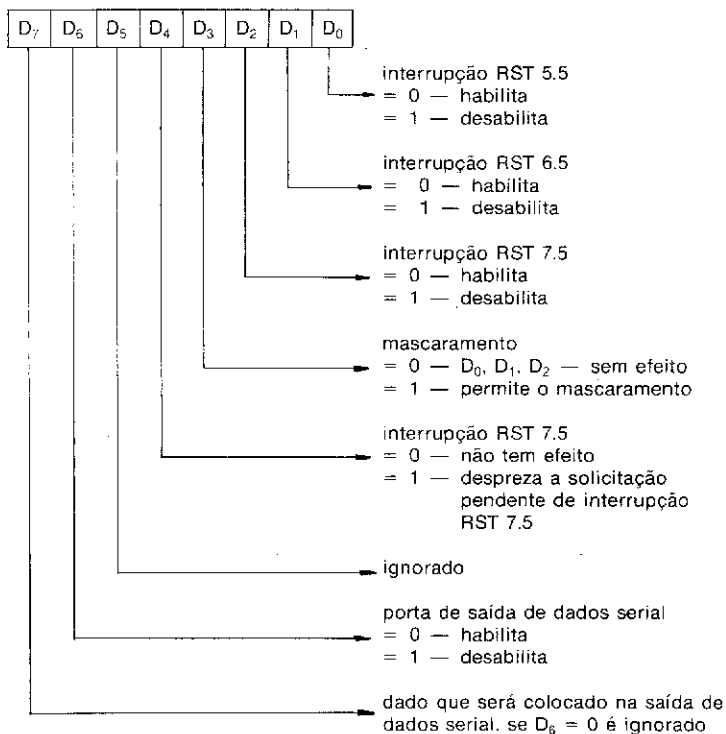
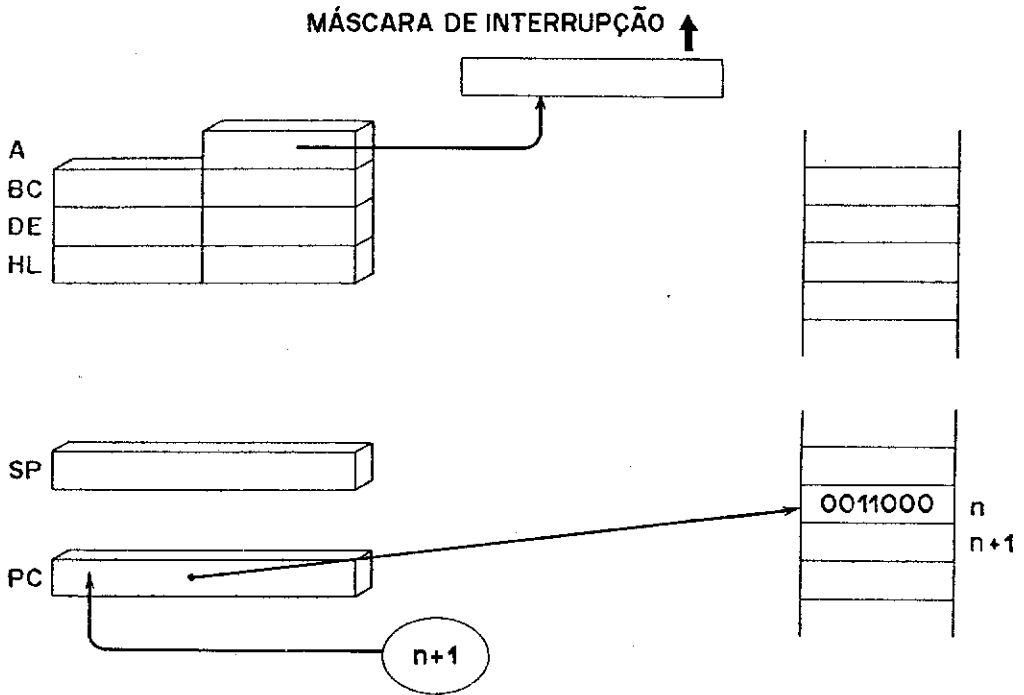
MÁSCARA DE INTERRUPÇÃO

A
BC
DE
HL

SP

PC

0011000
n
n+1

n+1

3.2. MICROPROCESSADOR Z80

O microprocessador Z80 da Zilog é um microprocessador de 8 bits, pertencente a uma família de componentes da Zilog. Ele é um microprocessador totalmente compatível em *software* com os microprocessadores da família 8080 Intel, com melhores implementações e maiores facilidades no *hardware*.

O Z80 possui uma arquitetura interna visível ao programador como mostra a Fig. 3.3. Como já dito acima, é um micro de 8 bits, com 16 bits para endereçamento (podendo endereçar até 64K de memória). Internamente possui 208 bits de memória R/W (RAM estática) que formam o conjunto de registradores.

É importante observar que apesar das instruções de máquina possuírem o mesmo código, a nível de programação assembler, os mnemônicos são diferentes.

	Conjunto Principal de Registradores		Conjunto Alternativo de Registradores	
Registradores de uso geral	Acumulador A	Flags F	Acumulador A'	Flags F'
	B	C	B'	C'
	D	E	D'	E'
	H	L	H'	L'

Vetor de Interrupção I	Refresh de Memória R	Registradores de Uso Especial
Registrador de Índice	IX	
Registrador de Índice	IY	
Apontador de Pilha	SP	
Apontador de Programa	PC	

Fig. 3.3 — Registradores do Z80.

Registradores

O Z80 possui 208 bits internos que compõem 18 registradores de 8 bits e quatro de 16 bits.

Na CPU existem dois acumuladores e dois registradores de flags associados. No Z80, diferentemente do 8080, há dois conjuntos de registradores, um principal e um alternativo, que não podem ser usados simultaneamente, mas é possível passar o uso de um para outro conjunto durante os programas. Daí existirem dois acumuladores e flags. Porém, o uso é independente. Para diferenciá-los serão chamados A, F e A' e F', bem como os registradores de uso geral.

Os acumuladores são registradores onde ocorrem todas as operações aritméticas e lógicas e os flags são bits de *status* das operações realizadas (resultado zero, overflow etc.). Para se trocar de um conjunto de registradores para o outro basta um comando de troca.

Registradores de Uso Especial

- PC — Apontador de Programa

O Apontador de Programa que guarda o endereço da instrução que está sendo executada é incrementado tão logo o conteúdo entra na barra de endereços, passando o PC para o endereço da próxima instrução. Em casos de desvios, o PC é alterado para o novo endereço.

- SP — Apontador de Pilha

O Apontador de Pilha contém o endereço atual do topo da pilha (16 bits), alocada em algum local da memória principal do computador. A pilha é uma área de memória onde são colocados os endereços quando há desvio ou interrupções. A pilha sempre decresce, isto é, o endereço diminui. Os dados são guardados do endereço mais alto para baixo. Durante a execução dos programas a pilha também pode servir para salvar dados.

- Registradores de Índice (IX e IY)

São dois registradores de 16 bits que são usados em modo de endereçamento indexados. Neste modo o registrador é usado como base para indicar uma região de memória de onde os dados serão tirados ou gravados. Um byte a mais, contendo o deslocamento, aparecerá sempre quando se usar estes registradores em instruções. Este deslocamento é especificado como um inteiro, servindo para endereçar tabelas de dados.

- Vetor de Interrupção (I)

A CPU Z80 pode chamar qualquer posição de memória em resposta a uma interrupção. O registrador I é usado para esta finalidade guardando os 8 bits de mais alta ordem do endereço, ficando os 8 de mais baixa ordem, sob a responsabilidade da unidade que interrompe. Isto permite que as rotinas de interrupção sejam colocadas em qualquer parte da memória com o tempo mínimo de acesso.

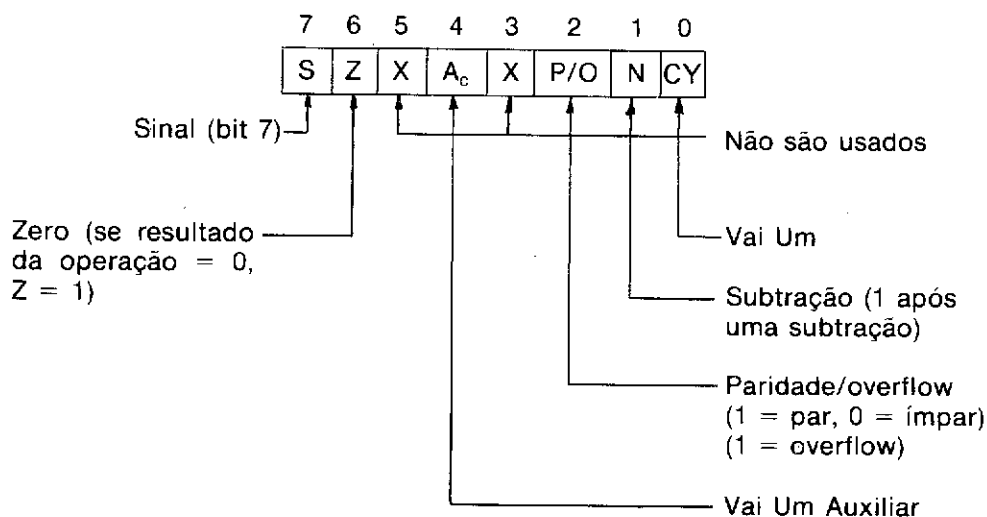
- Registrador de "Refresh" de Memória (R)

O registrador R é um contador de refresh para memórias dinâmicas. Este registrador é incrementado após cada "fetch" de instrução e controla o sinal de refresh. Estas operações são totalmente transparentes ao programador, que não precisa se preocupar com este registrador.

- Registradores de Uso Geral

Há dois conjuntos de registradores de uso geral. Estes registradores (B, C, D, E, H, L) podem ser usados separados ou formando pares, de 16 bits. Os dois conjuntos de registradores podem ser alternados no uso (nunca podem ser utilizados em conjunto) através de um único comando, o que permite, em interrupções, não ser preciso salvar o conteúdo dos registradores.

Flags

Como descrito acima, os indicadores (flags) são bits de *status* que são modificados ou não por cada instrução. O registrador F tem o seguinte formato:

S — (sinal) — Este bit coincide com o bit mais significativo do resultado e que, na notação complemento a 2, é zero para número positivo e 1 para número negativo.

Zero — (zero) — Indica que o resultado é zero (ou, no caso de uma comparação que os operandos são iguais).

A_c — ("Auxiliary Carry") — "VAI UM AUXILIAR" – Este bit indica a propagação do bit 3 para o bit 4 do resultado.

P/O — (Paridade ou Overflow) — Este bit tem dois usos distintos. Em operações aritméticas indica overflow. Para operações de entrada, rotação e lógicas indica a paridade (= 1, para paridade par; = 0 para paridade ímpar).

N — (subtração) — Este bit recebe o valor 1, após todas as operações de SUBTRAÇÃO, e o valor 0, após as operações de SOMA.

CY — ("Carry") — "VAI UM" — este bit indica a propagação a partir do bit mais significativo do acumulador.

Tipos de Endereçamento

Além dos métodos de endereçamento disponíveis no 8080, o Z80 dispõe de dois outros importantes tipos de endereçamento:

— INDEXADO
— RELATIVO

Indexado

O endereçamento indexado é utilizado quando o endereço do operando é obtido pela soma de um valor que consta da instrução com o conteúdo de um registrador especial, denominado Indexador. O Z80, dispõe de dois registradores indexadores, chamados IX e IY, e todas as referências a memória que podem ser feitas por HL, podem de forma alternativa ser feitas utilizando endereçamento indexado.

Por exemplo: ADD A, (IX + 10H) — com esta instrução, o acumulador será somado com o conteúdo da posição da memória fornecida pelo conteúdo do registrador IX, mais 10_{16}.

É importante observar que as instruções que utilizam endereçamento indexado, possuem dois bytes de código de instrução.

Relativo

O endereçamento relativo é uma forma de endereçar a memória, em relação ao valor atual do Apontador de Programa. O endereçamento pode ser feito no intervalo − 126 a + 129 bytes, a partir do valor do Apontador de Programa. As instruções de desvios relativos (JR, e JR condicionais) utilizam este tipo de endereçamento. Por exemplo: JR 3AH − o Apontador de Programa (PC) receberá como novo valor, o valor atual do PC mais o operando (3A). Observe que estas instruções necessitam de apenas 2 bytes, um para o código e outro para o endereço relativo.

Z80 × 8080A/8085

Código de Máquina Compatível

Todas as instruções de máquina 8080A estão disponíveis no Z80.

Todos os registradores disponíveis no 8080A/8085, também estão no Z80.

O Z80 dispõe de instruções, registros e outros recursos que não disponível no 8080A, e os programas em Z80 geralmente não são executáveis em 8080A.

Assembler

A compatibilidade de códigos de máquina não é estendida ao programa-fonte escrito em linguagem Assembler Z80, pois os mnemônicos utilizados são diferentes. A TABELA COMPARATIVA DE INSTRUÇÕES lista todos os mnemônicos Z80, com os seus correspondentes em 8080. Porém, nada impede que o programador use o Assembler 8080 para montar seus programas de Z80 desde que só sejam utilizadas as instruções do 8080. Tal prática é muito comum em Sistemas CP/M.

MNEMÔNICO 8080/8085	MNEMÔNICO Z80
ACI e	ADC A, e
ADC r	ADC A, r
ADC M	ADC A, (HL)
ADD r	ADD A, r
ADD M	ADD A, (HL)
ADI e	ADD A, e

MNEMÔNICO 8080/8085	MNEMÔNICO Z80
ANA r	AND r
ANA M	AND (HL)
ANI e	AND e
CALL end	CALL end
CC end	CALL C, end
CM end	CALL M, end
CMA	CPL
CMC	CCF
CMP r	CP (HL)
CMP M	CP (HL)
CNC end	CALL NC, end
CNZ end	CALL NZ, end
CP end	CALL P, end
CPE end	CALL PE, end
CPI e	CP e
CPO end	CALL PO, end
CZ end	CALL Z, end
DAA	DAA
DAD B [D; H ou SP]	ADD HL, BC [HL, DE; HL, HL; HL, SP]
DCR r	DEC r
DCR M	DEC (HL)
DCX B [D; H ou SP]	DEC BC [DE; HL ou SP]
DI	DI
EI	EI
HLT	HALT
IN porta	IN A, (porta)
INR r	INC r
INR M	INC (HL)
INX B [D; H ou SP]	INC BC [DE; HL ou SP]
JC end	JP C, end
JM end	JP M, end
JMP end	JP end
JNC end	JP NC, end
JP end	JP P, end
JNZ end	JP NZ, end
JPE end	JP PE, end
JPO end	JP PO, end
JZ end	JP Z, end
LDA end	LD A, (end)
LDAX B [D]	LD (BC) [(DE)]
LHLD end	LD HL, (end)
LXI B, e16 [D, e16; H, e16 ou SP, e16]	LD BC, e16 [DE, e16; HL, e16; SP, e16]
MOV r_1, r_2	LD r_1, r_2
MOV r, M	LD r, (HL)
MOV M, r	LD (HL), r
MVI r, e	LD r, e
MVI M, e	LD (HL), e
NOP	NOP
ORA r	OR r
ORA M	OR (HL)
ORI e	OR e
OUT porta	OUT (porta), A
PCHL	JP (HL)
POP B [D, H ou PSW]	POP BC [DE, HL ou AF]
PUSH B [D, H ou PSW]	PUSH BC [DE, HL ou AF]
RAL	RLA
RAR	RRA
RC	RET C
RET	RET

MNEMÔNICO 8080/8085	MNEMÔNICO Z80
RIM	—
RLC	RLCA
RM	RET M
RNC	RET NC
RNZ	RET NZ
RP	RET P
RPE	RET PE
RPO	RET PO
RRC	RRCA
RST n	RST n
RZ	RET Z
SBB r	SBC A, r
SBB M	SBC A, (HL)
SBI e	SBC A, e
SHLD end	LD (end), HL
SIM	—
SPHL	LD SP, HL
STA end	LD (end), A
STAX B [D]	LD (BC), A [(DE), A]
STC	SCF
SUB r	SUB r
SUB M	SUB (HL)
SUI e	SUB e
XCHG	EX DE, HL
XRA r	XOR r
XRA M	XOR (HL)
XRI e	XOR e
XTHL	EX (SP), HL

Códigos de Máquina de 2 bytes

Os códigos de máquina não utilizados pelo 8080A/8085, não são suficientes para implementar todas as instruções adicionais que o Z80 dispõe; deste modo é necessário que algumas instruções tenham 2 bytes para representar o código de operação. As instruções que têm o primeiro byte igual a CB, DD, ED, FD, necessitam de um segundo byte para caracterizar a operação a ser executada.

Flag P/O

O Z80 utiliza o flag P/O, para indicar overflow em operações aritméticas, e paridade em operações lógicas, enquanto o 8080A usa o flag para indicar apenas paridade.

Instrução DAA

Em Z80, esta instrução pode ser utilizada após uma soma ou subtração de dois números em BCD, enquanto em 8080/8085 só pode ser utilizada após uma soma.

Instruções de Rotação

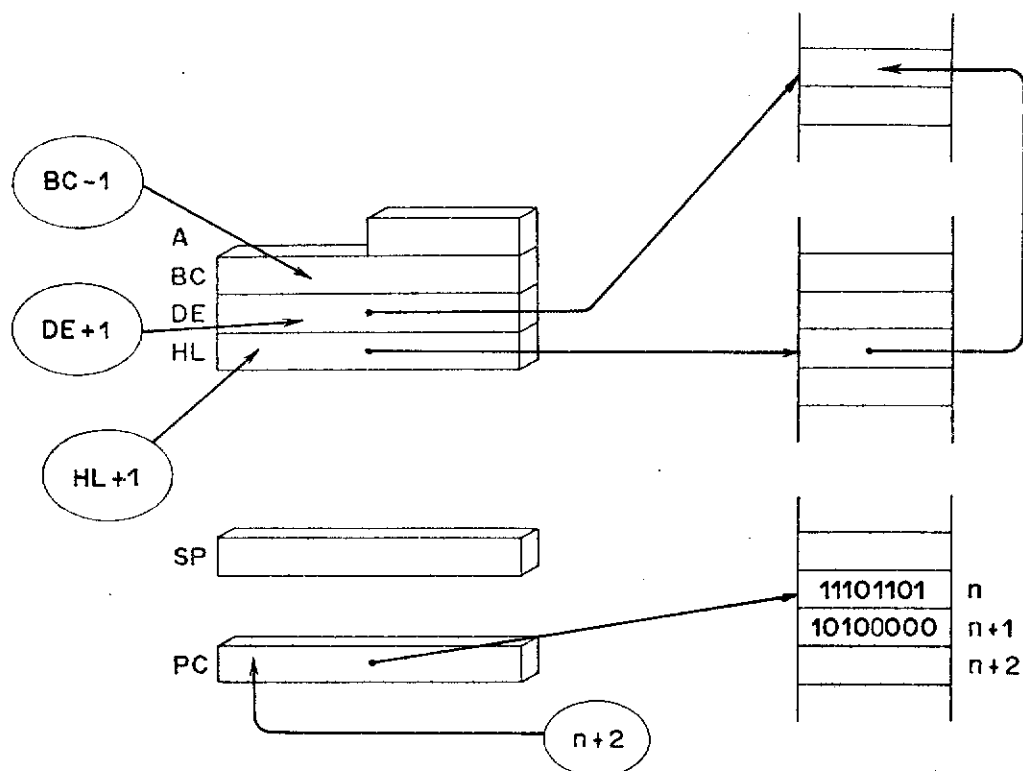
No Z80 as instruções de rotação fazem $A_c = 0$, enquanto no 8080A/8085 estas instruções não afetam o flag A_c.

Instruções RIM/SIM (8085)

O Z80 não é compatível com instruções RIM e SIM, disponíveis no 8085. Os códigos utilizados para RIM e SIM, são utilizados no Z80 para desvios relativos (JR NZ, JR NC, respectivamente).

ALGUMAS INSTRUÇÕES Z80

TRANSFERÊNCIA DE UM DADO ENTRE POSIÇÕES DE MEMÓRIA, INCREMENTANDO PONTEIROS DE ORIGEM E DESTINO — LDI

Consiste em transferir um dado de um byte que se encontra em uma posição de memória cujo endereço é indicado por HL para uma posição de memória cujo endereço é indicado por DE. DE e HL são incrementados. BC é decrementado.

FLAGS: $A_c = 0$, $N = 0$
Se $BC-1 \neq 0$ *então* P/O = 0
senão P/O = 1

Ex.: Suponha que BC contém $003B_{16}$, DE contém $312C_{16}$ e HL contém 1015_{16}, e a posição de memória 1015_{16} contém FF_{16}.

LDI 11101101 10100000 a posição de memória $312C_{16}$ passará a conter FF_{16}, BC conterá $003A_{16}$, DE conterá $312D_{16}$ e HL contém 1016_{16}

TRANSFERÊNCIA DE DADOS ENTRE POSIÇÕES DE MEMÓRIA, INDICADAS POR PONTEIROS DE ORIGEM E DESTINO ATÉ QUE O CONTADOR CHEGUE A ZERO (INCREMENTANDO PONTEIROS) — LDIR

Esta instrução é idêntica à LDI, exceto que a transferência é repetida, até que BC contenha zero. Após cada byte transferido, as interrupções são atendidas. Essa instrução é extremamente útil para transferência de blocos de dados de uma área de memória para outra.

FLAGS: $A_c = 0$, $N = 0$, P/O = 0

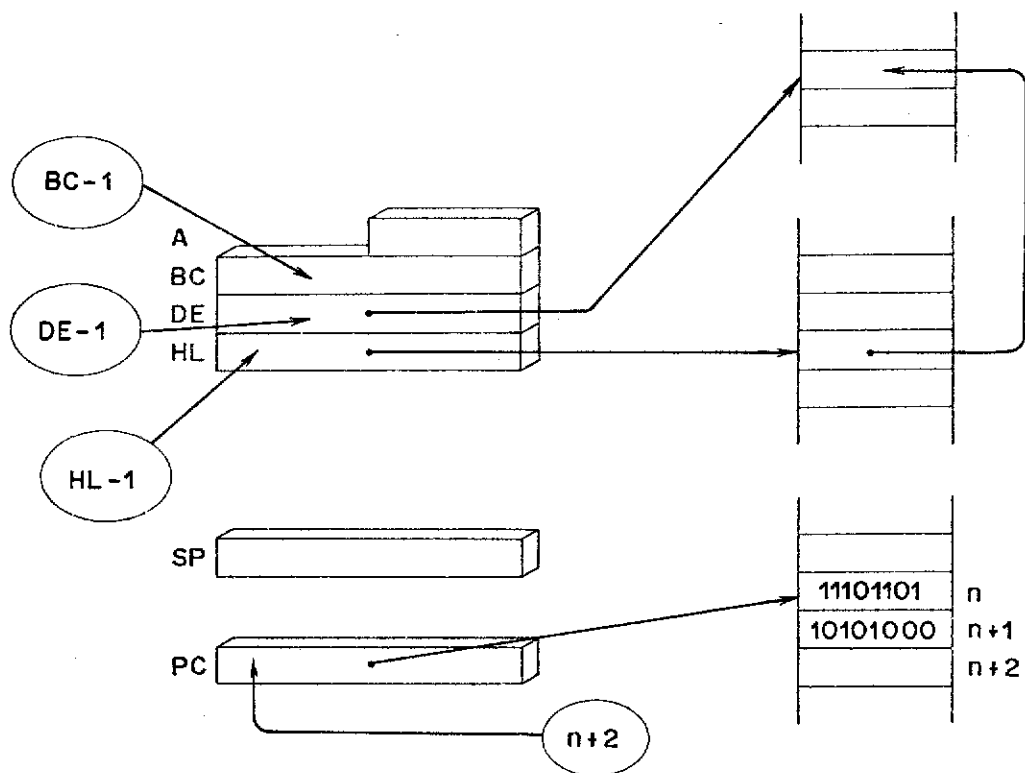
Ex.: Suponha os seguintes conteúdos de registradores e posições de memória:

HL = 1015_{16} $(1015_{16}) = 10_{16}$
DE = $312C_{16}$ $(1016_{16}) = 7A_{16}$
BC = 0003_{16} $(1017_{16}) = 3F_{16}$

LDIR 11101101 10110000 Após instrução teremos:

HL = 1018_{16} $(1015_{16}) = 10_{16}$ $(312C_{16}) = 10_{16}$
DE = $312F_{16}$ $(1016_{16}) = 7A_{16}$ $(312D_{16}) = 7A_{16}$
BC = 0000 $(1017_{16}) = 3F_{16}$ $(312E_{16}) = 3F_{16}$

TRANSFERÊNCIA DE UM DADO ENTRE POSIÇÕES DE MEMÓRIA, DECREMENTANDO PONTEIROS DE ORIGEM E DESTINO — LDD

Consiste em transferir um dado de um byte que se encontra em uma posição de memória cujo endereço é indicado por HL para uma posição de memória cujo endereço é indicado por DE. DE, HL e BC são decrementados.

FLAGS: $A_c = 0$; $N = 0$
Se BC−1 ≠ 0 *então* P/O = 0
senão P/O = 1

Ex.: Suponha os seguintes conteúdos de registradores e posições de memória:

$HL = 1015_{16}$ $(1015_{16}) = FF_{16}$
$DE = 312C_{16}$
$BC = 3B_{16}$

LDD 11101101 Após a execução da instrução teremos:

$HL = 1014_{16}$ $(1015_{16}) = FF_{16}$
$DE = 312B_{16}$ $(312C_{16}) = FF_{16}$
$BC = 3A_{16}$

TRANSFERÊNCIA DE DADOS ENTRE POSIÇÕES DE MEMÓRIA INDICADAS POR PONTEIROS DE ORIGEM E DESTINO, ATÉ QUE O CONTADOR CHEGUE A ZERO (DECREMENTANDO PONTEIROS) — LDDR

Esta instrução é idêntica à LDD, exceto que a transferência é repetida, até que BC contenha zero. Após cada byte transferido as interrupções são atendidas. Essa instrução é extremamente útil para transferência de blocos de dados de uma área de memória para outra.

FLAGS: $A_c = 0$; $N = 0$; $P/O = 0$

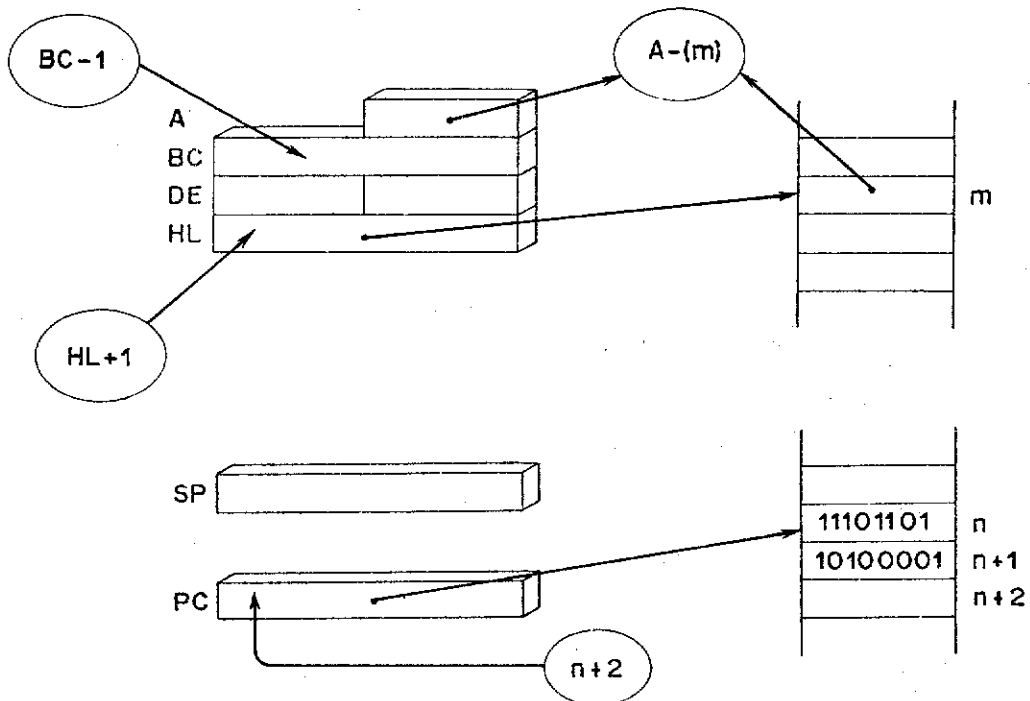
Ex.: Suponha os seguintes conteúdos de registrados e posições de memória:

$HL = 1015_{16}$ $(1013_{16}) = C1_{16}$
$DE = 312C_{16}$ $(1014_{16}) = B3_{16}$
$BC = 0003_{16}$ $(1015_{16}) = 10_{16}$

LDDR 11101101 10111000 Após a execução da instrução teremos:

$HL = 1012_{16}$ $(1013_{16}) = C1_{16}$ $(312A_{16}) = C1_{16}$
$DE = 3129_{16}$ $(1014_{16}) = B3_{16}$ $(312B_{16}) = B3_{16}$
$BC = 0000$ $(1015_{16}) = 10_{16}$ $(312C_{16}) = 10_{16}$

COMPARAÇÃO ENTRE O ACUMULADOR E UMA POSIÇÃO DE MEMÓRIA, DECREMENTANDO CONTADOR, E INCREMENTANDO PONTEIRO — CPI

Consiste em comparar o conteúdo do acumulador com o conteúdo da posição de memória, especificada por HL. Se forem iguais, o flag Z é ligado. HL é incrementado e BC (usado como contador) é decrementado.

FLAGS: S, Z, A_c
$N = 1$
Se $BC-1 \neq 0$ *então* $P/O = 0$
senão $P/O = 1$

Ex.: Suponha os seguintes conteúdos para registradores e posições de memória:

$A = E3_{16}$
$HL = 4000_{16}$
$BC = 0005_{16}$
$(4000_{16}) = E3_{16}$

CPI 11101101 10100001 Após a execução da instrução teremos:

$A = E3_{16}$
$HL = 4001_{16}$
$BC = 0004_{16}$
$Z = 1;\ A_c = 0;\ P/O = 0;\ N = 1$

COMPARAÇÃO ENTRE O ACUMULADOR E POSIÇÕES DE MEMÓRIA DECREMENTANDO CONTADOR, INCREMENTANDO PONTEIRO, ATÉ ENCONTRAR OU ATÉ QUE O CONTADOR CHEGUE A ZERO — CPIR

Esta instrução é idêntica à CPI, exceto que a comparação é executada até encontrar na memória uma posição com valor idêntico ao acumulador, ou até que o contador chegue a zero. Após cada comparação as interrupções são atendidas. Essa instrução é extremamente útil para executar a busca de um elemento em uma tabela.

FLAGS: S, Z, A
N = 1
Se BC−1 ≠ 0 *então* P/O = 0
senão P/O = 1

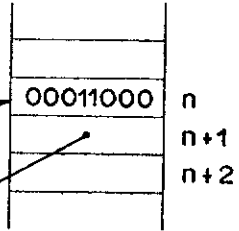
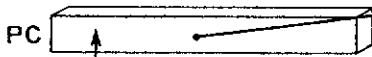
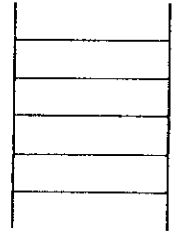
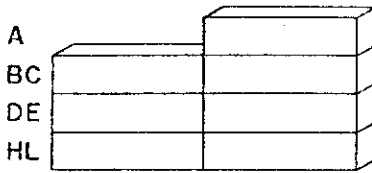
Ex.: Suponha os seguintes conteúdos para registradores e posições de memória:

$A = E3_{16}$ — $(4000_{16}) = F9_{16}$
$HL = 4000_{16}$ — $(4001_{16}) = 15_{16}$
$BC = 0005_{16}$ — $(4002_{16}) = AA_{16}$
$(4003_{16}) = E3_{16}$
$(4004_{16}) = B1_{16}$

CPIR 1110110 10110001 Após a execução teremos:

$A = E3_{16}$ — $(4000_{16}) = F9_{16}$
$HL = 4004_{16}$ — $(4001_{16}) = 15_{16}$
$BC = 0001_{16}$ — $(4002_{16}) = AA_{16}$
$(4003_{16}) = E3_{16}$
$(4004_{16}) = B1_{16}$
$Z = 1;\ A_c = 0;\ P/O = 0;\ N = 1$

DESVIO RELATIVO AO CONTADOR DE PROGRAMA — JR e

O conteúdo do segundo byte da instrução é somado ao valor atual do contador de programa (PC) mais 2. Este endereço é carregado no PC. O desvio relativo permite referenciar posições no intervalo −126 a 129 em relação ao PC.

A
BC
DE
HL
SP
PC
00011000 n
n+1
n+2
PC+(n+1)+2

FLAGS: ———

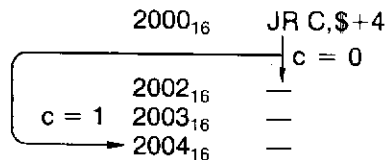
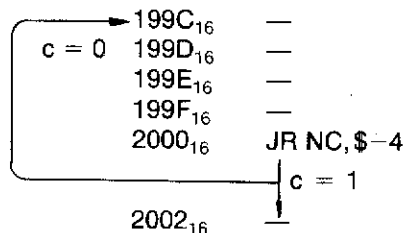
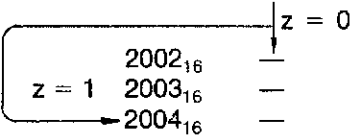
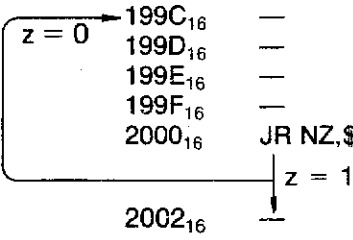
Ex.: Suponha que o PC aponta para 2000_{16}

JR \$+3 00011000 00000001 Após a instrução o novo PC será 4003_{16}

DESVIO RELATIVO AO CONTADOR DE PROGRAMA EM FUNÇÃO DOS FLAGS — JR NC,e; JR NZ,e; JR Z,e

As instruções são idênticas JR e, exceto que o desvio só é realizado, se a condição indicada for verdadeira; caso contrário, o programa segue o fluxo normal.

FLAGS: ———

Ex.:

TESTA BIT i no registro r — BIT i,r

O bit i do registro indicado é complementado e o resultado é colocado no flag Z, enquanto o bit i referenciado permanece inalterado.

S Z A_C P/O N CY

A
BC
DE
HL

bit xxx de yyy é complementado

SP

PC

n+2

10101011 n
01xxxyyy n+1
n+2

FLAGS: Z
N = 0; A_c = 1
S e P/O ficam indefinidos

```
       BIT                    i , r
11001011 00000001          01xxxyyy
```

bit testado		*registro*	
xxx : 000	0	yyy : 000	B
001	1	001	C
010	2	010	D
011	3	011	E
100	4	100	H
101	5	101	L
110	6	111	A
111	7		

Ex.: Suponha que o registro B contém 11110111

BIT 3,B 11001011 00000001 01011000 Após a execução da instrução teremos: Z = 1, enquanto o bit 3 de B permanecerá com zero

3.3. EXERCÍCIOS

1. Transcreva para binário as seguintes instruções Assembler 8085 (utilize o Apêndice A):

a) MOV D,A
b) ADI 31
c) RIM

2. Transcreva para binário as seguintes instruções Assembler Z80 (utilize o Apêndice B):

a) SBC A,$7F_{16}$
b) ADD A,31
c) SRL (IX+$3A_{16}$)

3. Utilizando os Apêndices A e B, determine a que instruções Assembler 8080/8085 e Z80 os códigos binários abaixo correspondem:

a) 01111110
b) 11101011
c) 00100111

4. Determine o conteúdo do acumulador e dos flags de condição de um microprocessador 8080/8085, após a execução de cada uma das instruções abaixo. Inicialmente o acumulador contém $B3_{16}$ e o flag VAI UM (CY) está ligado (1).

a) ANA A
b) ORA A
c) XRA A
d) ADI 96_{16}
e) SBI $OC9_{16}$
f) ACI 90_{16}
g) RRC
h) RAL

5. Em um sistema baseado em um microprocessador 8080/8085, temos as seguintes condições iniciais:

A = $4F_{16}$
B = 27_{16}
H = 00_{16}
L = OF_{16}

Memória
$(000F_{16}) = 2D_{16}$

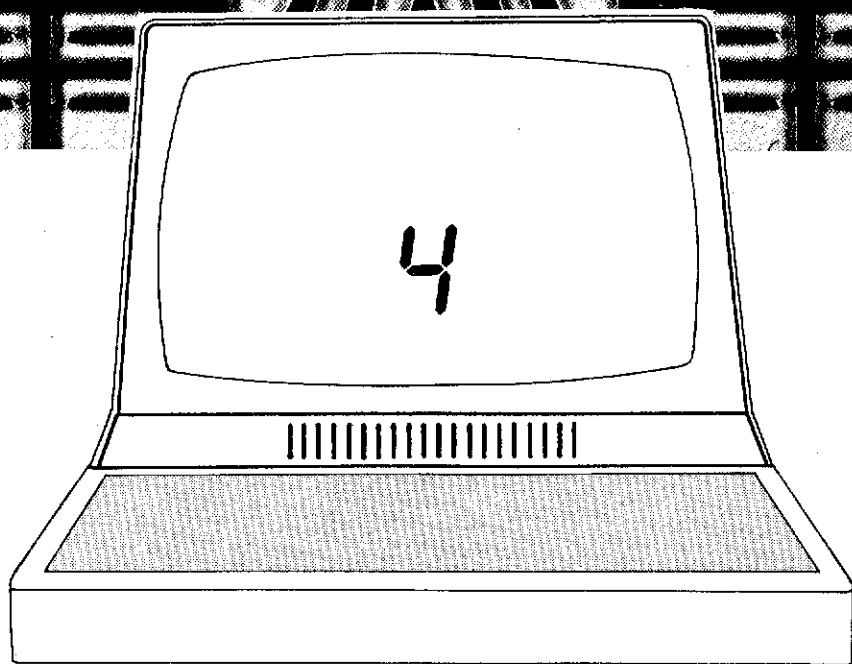
E na posição de memória 100_{16}, inicia o seguinte trecho do programa:

```
ADD B
ADC M
DAA
MOV M,A
```

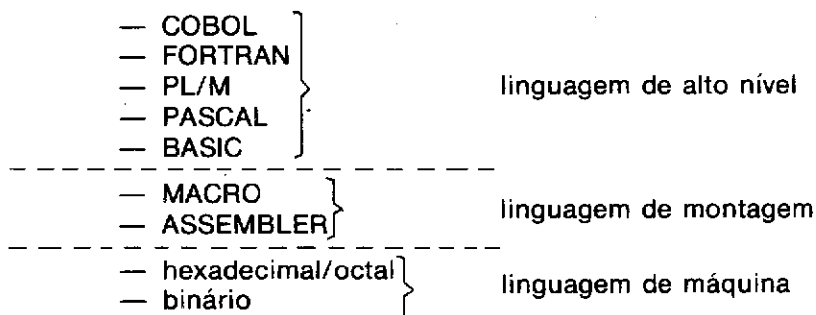
Após a execução do trecho acima, determine o conteúdo de todos os registradores e posições de memória que foram alterados, bem como o valor atual do Apontador de Programa (PC).

6. Utilizando as instruções ADD e ADC, escreva um trecho de programa em Assembler 8080/8085 que execute a soma de dois números de 24 bits (3 bytes consecutivos, sendo o primeiro o mais significativo). Os dois números estão armazenados em seis posições consecutivas a partir do endereço de memória 120_{16}. O resultado deverá ser armazenado nos endereços de memória 126_{16}, 127_{16} e 128_{16}.

7. Repita o exercício anterior para subtração (utilize as instruções SUB e SBB).

8. Na linguagem de programação FORTRAN, existe o comando IF(Y)100, 200, 300. Este comando provoca uma transferência do controle do programa para o comando 100, se $Y < 0$; para o comando 200, se $Y = 0$; e para o comando 300, se $Y > 0$. Assumindo que cada número de comando corresponde a um endereço de memória, e que Y é um valor de 8 bits em complemento a dois, armazenado na posição de memória 120_{16}, escreva em Assembler 8080/8085 um trecho de programa equivalente ao comando IF do FORTRAN.

PROGRAMAÇÃO

4.1. DESENVOLVIMENTO DE PROGRAMAS

Para elaboração de programas para micro-sistemas podemos definir 3 níveis possíveis de programação.

Linguagem de Máquina

A utilização de uma linguagem de máquina é a maneira mais elementar de se apresentar um programa. O conjunto de códigos binários que compõe o programa, identifica ações, que são decodificadas e executadas pelo microprocessador.

A programação direta em linguagem de máquina é trabalhosa, e sujeita a muitos erros em função da manipulação de números binários.

O exemplo a seguir apresenta um programa em linguagem de máquina.

```
0010 0001
0000 1101
0000 0001
0111 1110
0010 0011
1011 1110
1101 0010
0000 1010
0000 0001
0111 1110
0010 0011
0111 0111
0111 0110
0111 1011
0011 1010
0000 0000
```

Observe que entender o que o programa faz é bastante difícil, além de que, se trocarmos apenas um dos dígitos binários utilizados o programa possivelmente provocará um erro de execução difícil de ser detetado.

Além da programação ser bastante trabalhosa, é necessário carregar o programa na memória do micro-sistema, para que o programa seja efetivamente executado. A carga do programa é normalmente feita por um programa denominado "bootstrap", que fica residente em uma memória tipo ROM; este programa é responsável pela entrada do programa de um meio de entrada para memória RAM do micro-sistema. O armazenamento do programa "bootstrap", ou outra aplicação que necessite ser armazenado em ROM, é feito por um equipamento denominado Programador de ROM.

Para facilitar um pouco a representação de programas em linguagem de máquina, é usual que se trabalhe com uma codificação hexadecimal, que permite reduzir cada 4 dígitos binários para apenas um dígito hexadecimal. Visto que normalmente as palavras dos microprocessadores são de 1 byte (8 bits), passamos a representar cada palavra com apenas 2 dígitos hexadecimais, facilitando portanto a manipulação dos códigos.

O programa anterior passará a ser representado por:

```
21
0D
01
7E
23
BE
D2
0A
01
7E
23
77
76
7B
3A
00
```

A programação em linguagem de máquina é normalmente utilizada em sistemas de baixo custo, que não dispõem de recursos de suporte à programação. Este esquema só é exeqüível se o programa for composto de um pequeno número de instruções (entre 10 e 150, por exemplo). Para programas maiores este esquema é tedioso e sujeito a muitos erros, sendo necessário o desenvolvimento do programa em um outro equipamento com mais recursos.

Linguagem de Montagem

Quando um Montador Assembler está disponível em um micro-sistema, o programador fornece o programa em uma linguagem mnemônica. O programa Montador, lê os comandos assembler e traduz para o código de máquina correspondente. Quando um bom Montador é utilizado ele fornece uma série

de funções, com o objetivo de facilitar a programação. Normalmente é possível utilizar símbolos para representar valores, e estão disponíveis pseudo-instruções para dar valores aos símbolos. As instruções podem fazer referência a endereços através de símbolos.

O programa anterior seria escrito pelo programador da seguinte forma:

```
LXI   H, 10DH
MOV   A, M
INX   H
CMP   M
JNC   10AH
MOV   A, M
INX   H
MOV   M, A
HLT
```

No exemplo estamos utilizando mnemônicos válidos para a linguagem Assembler Intel 8080/8085.

Linguagem de Alto Nível

O programa pode também ser escrito em uma linguagem como BASIC, PASCAL, FORTRAN e outras.

As linguagens de alto nível possuem comandos poderosos que tornam a atividade de programação fácil e rápida.

Estes programas escritos nestas linguagens são traduzidos, gerando um complexo programa em linguagem de máquina que será executado posteriormente.

O programa responsável por esta tradução é normalmente denominado COMPILADOR ou INTERPRETADOR.

Ele é chamado de COMPILADOR, quando o programa é totalmente traduzido para linguagem de máquina, produzindo um código seqüencial, denominado código-objeto, e em uma fase posterior este código-objeto é carregado na memória do sistema, e o programa é executado.

Chamamos de INTERPRETADOR, quando os comandos da linguagem de alto nível são executados passo a passo, isto é, um comando de alto nível é traduzido para linguagem de máquina e em seguida o código resultante é executado.

Os INTERPRETADORES oferecem a vantagem de permitir uma execução interativa do programa, porém apresentam baixa eficiência em relação aos compiladores.

A seguir temos um programa escrito em BASIC, equivalente ao programa anterior.

```
10 INPUT PRIMEIRO,SEGUNDO
20 IF PRIMEIRO = SEGUNDO THEN PRINT "ELES SAO IGUAIS":END
30 IF PRIMEIRO > SEGUNDO THEN PRINT "O PRIMEIRO NUMERO E' MAIOR":END
40 PRINT "O SEGUNDO NUMERO E' MAIOR"
50 END
```

4.2. ASSEMBLER PARA MICROPROCESSADORES 8080/8085 E Z80

Formato

Um programa Assembler consiste de uma seqüência de comandos com o seguinte formato:

comando	RÓTULO:	OPERAÇÃO	OPERANDO	;COMENTÁRIOS

Usualmente o formato é livre sendo cada campo reconhecido pela sua posição.

Rótulo. Este campo é opcional, e é normalmente utilizado quando desejamos fazer uma referência simbólica a este comando.

Os rótulos são identificadores formados por caracteres alfanuméricos (alfabéticos e números), sendo o primeiro caracter necessariamente alfabético.

O rótulo quando existir pode opcionalmente ser seguido por um ":".

Entre os motivos que podemos citar para o uso de rótulos, temos:

— Um rótulo marca facilmente uma localização no programa.

— Quando um rótulo é trocado de local dentro do programa, não é necessário se preocupar com todas as referências a este rótulo, visto que o Montador Assembler, quando da montagem do programa, se encarregará das trocas necessárias.

— Não é necessário se preocupar com o endereçamento da memória, que é bastante útil, visto que muitos microprocessadores possuem instruções de tamanhos diferentes.

Operação. O campo OPERAÇÃO contém um mnemônico correspondente a uma instrução de máquina ou uma pseudo-instrução. As pseudo-instruções mais usuais estão descritas adiante.

Operando. O campo do operando, em geral, contém uma expressão composta de constantes e identificadores (rótulo) ou palavras reservadas.

A combinação dos elementos da expressão produz um valor de 16 bits, que é obtido em fase de montagem. Porém, como existem instruções que exigem um operando de 8 bits, apenas os bits de menor significância são utilizados.

Constantes

As constantes utilizadas podem ser apresentadas em várias bases. A base é indicada por um sufixo que segue a constante; os indicadores normalmente utilizados são B, para constantes binárias (base 2); O e Q para constantes octais (base 8); D, para constantes decimais (base 10) e H para constantes hexadecimais (base 16). A ausência de sufixo indica normalmente que a constante está na base 10.

7BH; 1111011B; 373Q; são possíveis representações da mesma constante que na base 10 é: 123.

Constantes do tipo "string" também podem ser representadas; elas são compostas por uma seqüência de caracteres ASCII, delimitados pelo símbolo (').

'A'
'ENTRE COM OS DADOS'
'ESCOLHA O PROGRAMA 1, 2 OU 3'

Palavras Reservadas

A utilização de palavras reservadas, tais como os registradores, no caso 8080, permitem que códigos de máquinas possam ser produzidos, a partir de informações complementares disponíveis no operando. Por exemplo, na instrução MOV, parte do código de máquina é fornecido no operando, isto é, a origem e o destino da operação de movimentação. Por exemplo; as palavras reservadas abaixo assumem os seguintes valores:

A — 7
B — 0
C — 1
D — 2
E — 3
H — 4
L — 5
M — 6
SP — 6
PSW — 6

Identificadores

O identificador corresponde ao endereço onde aparece como rótulo; em particular, se o rótulo precede uma pseudo-instrução do tipo EQUATE, o identificador é substituído pelo valor do operando correspondente.

```
VOLTA:  LXI   H,100H
        ___
        ___
        ___
        JMP   VOLTA
```

Na instrução JMP, VOLTA equivale ao endereço onde se encontra a instrução LXI H,100H.

As constantes, palavras reservadas, identificadores podem ser combinadas em expressões tais como:

10H + 37Q
('A' AND 5FH) + 'O'
('B' + B) OR PSW

Pseudo-Instruções

As pseudo-instruções não geram código de máquina, elas são utilizadas durante o programa para complementar as informações que permitam a montagem efetiva do programa, tais como: indicar o endereço inicial de carga do programa, reservar área de dados, definir equivalência entre identificadores e valores. A seguir descreveremos algumas das pseudo-instruções disponíveis:

ORG. Indica ao Montador ASSEMBLER o endereço de memória onde deverá ser iniciada a montagem do programa. A expressão do operando indica o endereço de início do programa. Um programa pode ter vários ORGs, e o Montador Assembler não verifica se existe superposição de áreas do programa.

Quando programas são rodados sob CP/M, podemos observar que muitos deles começam com ORG 100H; isto provoca que a geração de código comece na área do sistema reservada para programas (TPA — TRANSIENT PROGRAM AREA).

```
PC      CÓDIGO-OBJETO        PROGRAMA-FONTE
                             ORG     100H
0100    210D01               LXI     H,10DH
0103    7E
 .       .                    .
 .       .                    .
 .       .                    .
```

END. A pseudo END é opcional em programas escritos em assembler, mas quando usada indica o fim do programa, deve ser o último comando do programa (todos os demais comandos assembler subseqüentes são ignorados).

Alguns montadores permitem que o operando da pseudo END, forneça o endereço de execução do programa. Este endereço corresponde ao último registro de um arquivo que contém um programa-objeto.

EQU. A pseudo EQU é utilizada para fornecer um sinônimo para um valor numérico em particular. A sua forma usual é:

< identificador > EQU < expressão >

A expressão é calculada, e sempre que o identificador utilizado no campo de rótulo aparecer em um campo operando, o operando é substituído pelo valor correspondente.

Este recurso permite utilizar identificadores para representar valores que sejam utilizados com freqüência pelo programador. Suponha o seguinte exemplo:

```
TTYBASE     EQU    10H             ;PORTA E/S
TTYIN       EQU    TTYBASE;        ;ENTRADA PELA TTY
TTYOUT      EQU    TTYBASE + 1     ;SAÍDA PELA TTY
```

Em outros pontos do programa poderiam ter instruções do tipo:

```
IN      TTYIN       ; LÊ DA TTY PARA O REG A
---
OUT     TTYOUT      ; ESCREVE O REG A NA TTY
```

ao invés:

```
IN       10H
---
OUT      11H
```

DB. A pseudo DB permite definir e inicializar áreas de um byte. O formato do comando é:

<identificador> DB <exp1>, <exp2>, . . ., <expn>

As expressões devem indicar valores correspondentes a 1 byte, ou cadeia de caracteres ASCII.

As expressões são calculadas e colocadas em posições seqüenciais no código objeto. De forma similar os caracteres da cadeia ASCII são colocados em posições contíguas do código-objeto.

```
Exemplo:    DADOS:  DB  1,3,5,7
                    DB  377Q,1+3+5,0FFH
            MENS:   DB  'ENTRE COM O SEU NOME',0DH,0AH,'$'
```

As definições acima gerariam o seguinte código-registro:

```
01030507
FF09FF
454E54524520434F4D204F20534555204E4F4D450D0A24
```

DW. A pseudo DW é equivalente à DB, sendo que cada expressão é colocada em dois bytes. O formato é o seguinte:

<identificador> DW <exp1>, <exp2>, . . ., <expr>

Cada expressão do campo de operando é calculada em 16 bits (2 bytes), porém o armazenamento no código-objeto é feito de forma consistente com o microprocessador 8080, isto é, os 8 bits de menor significância são armazenados primeiro, seguidos dos 8 bits de maior significância.

```
Exemplo:    DADOS:    DW    3, 3AB0H, FF00H
```

O código-objeto correspondente seria:

```
0300B53A00FF
```

DS. A pseudo DS permite reservar uma área de memória, sem que a mesma seja inicializada.

O Montador Assembler continua a gerar o código-objeto após a área reservada pela pseudo DS.

Assembler Condicional

O Assembler Condicional é uma facilidade que permite ao programador desenvolver programas gerais que se adaptam a cada problema específico. Durante uma montagem do programa em função de certos parâmetros apenas alguns trechos do programa são efetivamente montados.

Em uma possível aplicação que em determinada parte do programa tenha opção de utilizar algoritmos diferentes em função das características do usuário, o Assembler Condicional pode ser usado para garantir que cada usuário tenha apenas o código que efetivamente precisa.

Para utilização de Assembler Condicional normalmente estão disponíveis pseudos do tipo: SET e IF/ENDIF.

SET. A pseudo SET é similar à pseudo EQU, sendo que rótulo utilizado pode aparecer mais de uma vez no mesmo programa.

Esta pseudo permite definir novos valores para um identificador em vários pontos do programa.

< identificador > SET < expressão >

IF/ENDIF. As pseudos IF/ENDIF definem um trecho que deve ou ser incluído ou excluído de um processo de Montagem.

```
IF < expressão >
———
———trecho em Assembler
———
———
ENDIF
```

Quando a pseudo IF é encontrada, a expressão é calculada, e caso o valor obtido for diferente de zero o trecho entre IF e a pseudo ENDIF é montado. Caso o valor obtido seja zero o trecho é apenas listado mas não montado.

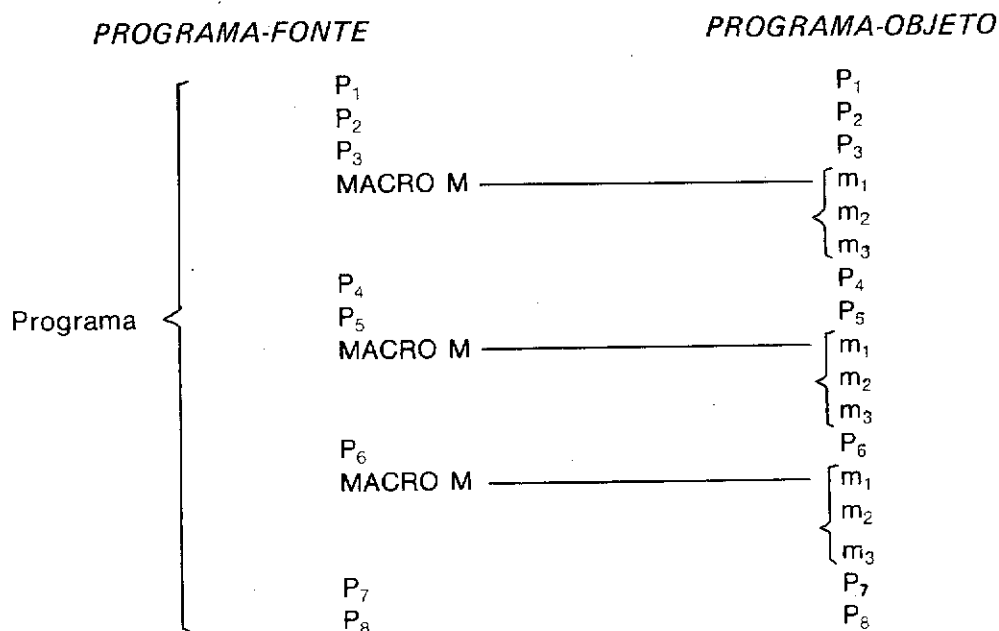
Infelizmente o Assembler Condicional tende a confundir o programa e torna difícil a sua leitura, portanto, só é recomendável que seja utilizado quando for extremamente necessário.

Macros

Freqüentemente uma seqüência de instruções ocorre muitas vezes em um programa. Repetir esta seqüência pode refletir uma necessidade da própria lógica do programa, ou então compensar alguma deficiência do conjunto de instruções do microprocessador utilizado.

Uma maneira de definir esta seqüência de instruções é através de MACROS, e sempre onde houver necessidade de repetir a seqüência, é feita uma referência à macro correspondente.

O Montador Assembler substituirá a referência à macro pela seqüência de instruções que ela representa.

	PROGRAMA-FONTE	*PROGRAMA-OBJETO*
Definição da Macro	*INÍCIO DA MACRO* M	
	m_1	
	m_2	
	m_3	
	FIM DA MACRO M	

As macros não são equivalentes às sub-rotinas, visto que o código-objeto é expandido, enquanto as sub-rotinas fazem parte do programa e em tempo de execução é executado um desvio para o trecho de programa onde está localizada a sub-rotina.

O uso de macros permite codificar programas-fontes mais curtos e melhor documentados, e é fácil alterá-las, bastando modificar a definição sem se preocupar onde ela é referenciada.

É importante observar que macros geram várias instruções fazendo o código do programa aumentar; além disso, as macros podem fazer uso de registradores e flags, podendo não ficar claro a sua utilização no corpo do programa.

4.3. EXEMPLOS DE PROGRAMAS EM ASSEMBLER 8080/8085

Maior entre Dois Números

Considere que duas posições consecutivas de memória contêm dois números sem sinal. Determine qual deles é o maior, e armazene na posição subseqüente.

ANTES	*DEPOIS*
(010D) = 7B	(010D) = 7B
(010E) = 3A	(010E) = 3A
	(010F) = 7B

```
                    ;*****************
                    ;*MAIOR DE 2 NUMEROS
                    ;*****************
0100                         ORG     100H
0100 210D01                  LXI     H,10DH
0103 7E                      MOV     A,M        ;PEGA PRIMEIRO NUMERO
0104 23                      INX     H
0105 BE                      CMP     M          ;COMPARA COM SEGUNDO NUMERO
0106 D20A01                  JNC     GUARDA
0109 7E                      MOV     A,M        ;O SEGUNDO NUMERO E' MAIOR
010A 23             GUARDA:  INX     H
010B 77                      MOV     M,A        ;GUARDA O MAIOR NUMERO
010C 76                      HLT
                    ;**************
                    ;*AREAS DE DADOS
                    ;**************
010D 7B                      DB      7BH        ;PRIMEIRO NUMERO
010E 3A                      DB      3AH        ;SEGUNDO NUMERO
010F 00                      DB      0          ;MAIOR
0110                         END
```

DUMP antes da execução:

```
0100 21 0D 01 7E 23 BE D2 0A 01 7E 23 77 76 7B 3A 00 !..~#....~#wv{:.
```

DUMP depois da execução:

```
0100 21 0D 01 7E 23 BE D2 0A 01 7E 23 77 76 7B 3A 7B !..~#....~#wv{:{
```

No exemplo acima, temos o relatório obtido pela montagem de um programa em um Assembler 8080/8085. Observe que à esquerda aparecem os endereços onde as instruções estarão armazenadas na memória, bem como o código de máquina, representado em dígitos hexadecimais, correspondente a cada instrução assembler. Todo o texto que segue o caracter ";" é encarado pelo Montador Assembler como comentário.

Como foi visto anteriormente, a pseudo-instrução ORG, redefine o endereço de carga do programa, no exemplo ORG 100H, faz com que a primeira instrução do programa LXI H, 100 H, seja carregada na posição 100_{16}.

Na instrução JNC GUARDA, no código de máquina gerado, o identificador GUARDA é substituído pelo endereço da instrução que é precedida pelo referido rótulo, no caso, o endereço é 010A, que codificado na instrução JNC(D2), como 0A01, conforme explicado na descrição da instrução de desvio.

As pseudo DB foram utilizadas para reservar posições de memória e definir valores que serão armazenados nestas posições.

Para ilustrar a execução do programa foi mostrado um "dump" de memória antes e depois da execução do programa. O "dump" é um relatório que mostra os conteúdos de um trecho da memória.

O "dump" está apresentado no seguinte formato:

```
0100 21 0D 01 7E 23 BE D2 0A 01 7E 23 77 76 7B 3A 7B !..~#....~#wv{:{
```

posição de memória	conteúdo de 16 posições de memória a partir de 100_{16} representado em dígitos hexadecimal	representação equivalente em códigos ASCII

Soma de Uma Série de Números

Uma série de números encontra-se em uma área de memória, precedida pelo número de elementos que compõem a série. Calcule a soma de todos os números e armazene o resultado em uma área de 16 bits sendo os oito primeiros, os menos significativos.

ANTES	*DEPOIS*
(0118) = 04	(0118) = 04
(0119) = BA	(0119) = BA
(011A) = 07	(011A) = 07
(011B) = F1	(011B) = F1
(011C) = 30	(011C) = 30
	(011B) = E2
	(011E) = 01

```
                   ;****************************
                   ;*SOMA DE UMA SERIE DE NUMEROS
                   ;****************************
0100                      ORG     100H
0100 211801               LXI     H,118H
0103 46                   MOV     B,M        ;B E' CONTADOR
0104 110000               LXI     D,0        ;DE E' O SOMATORIO DE 16 BITS
0107 23            VOLTA: INX     H
0108 7E                   MOV     A,M
                   ;*************
                   ;*SOMA A COM DE
                   ;*************
0109 83                   ADD     E
010A 5F                   MOV     E,A
010B D20F01               JNC     PULA
010E 14                   INR     D          ;SE CY=1
010F 05            PULA:  DCR     B          ;SE CY=0 DECREMENTA O CONTADOR
0110 C20701               JNZ     VOLTA
                   ;*********************
                   ;*GUARDANDO O RESULTADO
                   ;*********************
0113 EB                   XCHG               ;COLOCO O RESULTADO EM HL
0114 221D01               SHLD    RESULT     ;GUARDO O RESULTADO
0117 76                   HLT
                   ;*************
                   ;AREAS DE DADOS
                   ;*************
0118 04                   DB      04         ;NUMERO DE ELEMENTOS
0119 BA07F130             DB      0BAH,07H,0F1H,30H    ;NUMEROS A SEREM SOMADOS
011D 0000          RESULT: DW     0
011F                      END
```

DUMP antes da execução:

```
0100 21 18 01 46 11 00 00 23 7E 83 5F D2 0F 01 14 05 !..F...#~._.....
0110 C2 07 01 EB 22 1D 01 76 04 BA 07 F1 30 00 00 00 ...."..v....0...
```

DUMP depois da execução:

```
0100 21 18 01 46 11 00 00 23 7E 83 5F D2 0F 01 14 05 !..F...#~._.....
0110 C2 07 01 EB 22 1D 01 76 04 BA 07 F1 30 E2 01 00 ...."..v....0...
```

Cálculo de CHECKSUM

CHECKSUM é um método freqüentemente usado na transferência de dados de uma memória auxiliar (p. ex. fita cassete) para memória principal, e vice-versa, com o objetivo de detetar erros durante o processo de transferência. O método consiste em adicionar a uma cadeia de bytes, um byte adicional denominado CHECKSUM, quando esta mesma cadeia for utilizada, um novo byte de CHECKSUM é calculado e comparado com o que foi acrescentado à cadeia. Uma maneira usual de obter o CHECKSUM de uma cadeia de bytes é calcular um "ou-exclusivo" de todos os bytes.

ANTES	*DEPOIS*
(0111) = 04	(0111) = 04
(0112) = 15	(0112) = 15
(0113) = F0	(0113) = F0
(0114) = 44	(0114) = 44
(0115) = 03	(0115) = 03
	(0113) = A2

15 ⩡ F0 ⩡ 44 ⩡ 03

00010101 ⩡ 11110000 ⩡ 01000100 ⩡ 00000011 = 10100010

BYTE DE CHECKSUM = A2

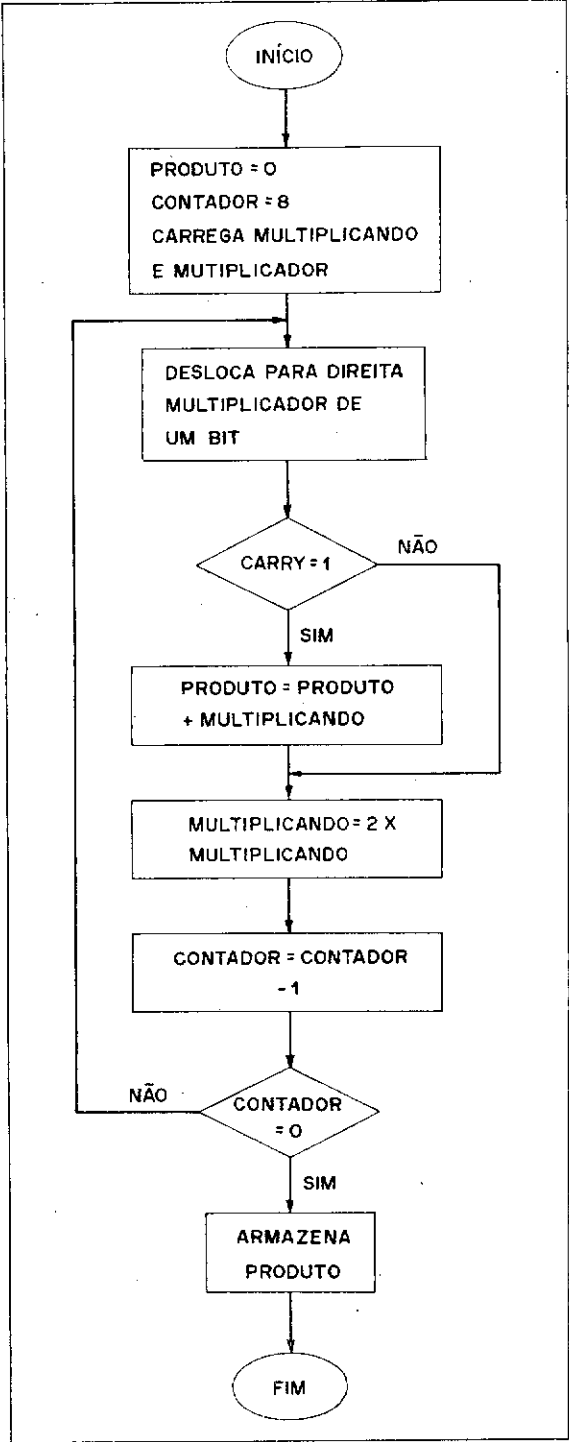
```
                              ;*********
                              ;*CHECKSUM
                              ;*********
0100                          ORG     100H
0100 211101                   LXI     H,0111H
0103 4E                       MOV     C,M       ;RECEBE NUMERO DE DADOS
0104 23                       INX     H
0105 7E                       MOV     A,M       ;PEGA PRIMEIRO NUMERO
0106 0D                       DCR     C         ;DECREMENTA CONTADOR
0107 23             VOLTA:    INX     H
0108 AE                       XRA     M
0109 0D                       DCR     C         ;DECREMENTA CONTADOR
010A C20701                   JNZ     VOLTA
010D 321601                   STA     0116H     ;CARREGA O RESULTADO
0110 76                       HLT
0111 04                       DB      04        ;NUMERO DE DADOS
0112 15F04403                 DB      15H,0F0H,44H,03H       ;DADOS
0116 00                       DB      0         ;CHECKSUM
0117                          END
```

DUMP antes da execução:

```
0100 21 11 01 4E 23 7E 0D 23 AE 0D C2 07 01 32 16 01 !..N#~.#.....2..
0110 76 04 15 F0 44 03 00 00 v...D...
```

DUMP depois da execução:

```
0100 21 11 01 4E 23 7E 0D 23 AE 0D C2 07 01 32 16 01 !..N#~.#.....2..
0110 76 04 15 F0 44 03 A2 00 v...D...
```

Fluxograma: multiplicação de dois números

Multiplicação de Dois Números

Considere dois números de oito bits, sem sinal, armazenado em posições consecutivas de memória. Calcule o produto, e armazene o resultado em uma área de 16 bits, sendo os oito primeiros bits os menos significativos.

Para elaborarmos o programa para executar a multiplicação podemos analisar o método que adotamos quando fazemos uma multiplicação à mão. Como os números são binários apenas multiplicamos por 0 ou por 1. Quando multiplicamos por zero, obviamente o resultado é zero, quando multiplicamos por 1 o resultado é o multiplicando.

```
Multiplicando: 43              01000011
Multiplicador: 3A          ×   00111010
                               --------
                               00000000
                              01000011
                             00000000
                            01000011
                           01000011
                          01000011
                         00000000
                        00000000
                        ---------------
                        000111100101110
```

ANTES	*DEPOIS*
(0123) = 43	(0123) = 43
(0124) = 3A	(0124) = 3A
	(0125) = 2E
	(0126) = 07

```
                   ;******************************
                   ;*MULTIPLICACAO DE DOIS NUMEROS
                   ;******************************
0100                      ORG     100H
0100 212301               LXI     H,AREA
0103 4E                   MOV     C,M       ;CARREGA MULTIPLICANDO
0104 0600                 MVI     B,0       ;ZERA PARTE ALTA DO MULTIPLICANDO
0106 23                   INX     H
0107 7E                   MOV     A,M       ;CARREGA MULTIPLICADOR
0108 210000               LXI     H,0       ;HL RECEBARA' O PRODUTO
010B 1608                 MVI     D,8       ;INICIALIZO CONTADOR DE BITS
010D 0F            VOLTA: RRC               ;DESLOCO PARA DIREITA O MULTIPLICADOR
010E D21201               JNC     PULA1
0111 09                   DAD     B
                   ;******************************
                   ;*MULTIPLICO MULTIPLICANDO POR 2
                   ;******************************
0112 5F            PULA1: MOV     E,A       ;SALVO EM E VALOR DO MULTIPLICADOR
0113 79                   MOV     A,C
0114 A7                   ANA     A         ;ZERO CY
0115 17                   RAL               ;MULTIPLICANDO X 2
0116 4F                   MOV     C,A
0117 78                   MOV     A,B       ;AGORA VOU FAZER ROTATE EM B
0118 17                   RAL
0119 47                   MOV     B,A

011A 7B                   MOV     A,E       ;RETORNO VALOR DO MULTIPLICADOR
011B 15                   DCR     D         ;DECREMENTO CONTADOR
011C C20D01               JNZ     VOLTA
```

```
                 ;*****************
                 ;*ARMAZENA PRODUTO
                 ;*****************
011F 222501              SHLD    PROD
0122 76                  HLT
                 ;***************
                 ;*AREAS DE DADOS
                 ;***************
0123 43          AREA:   DB      43H     ;MULTIPLICANDO
0124 3A                  DB      3AH     ;MULTIPLICADOR
0125 0000        PROD:   DW      0
0127                     END
```

DUMP antes da execução:

```
0100 21 23 01 4E 06 00 23 7E 21 00 00 16 08 0F D2 12 !#.N..#~!.......
0110 01 09 5F 79 A7 17 4F 78 17 47 78 15 C2 0D 01 22 ..-y..Ox.G(...."
0120 25 01 76 43 3A 00 00 00 %.vC:...
```

DUMP depois da execução:

```
0100 21 23 01 4E 06 00 23 7E 21 00 00 16 08 0F D2 12 !#.N..#~!.......
0110 01 09 5F 79 A7 17 4F 78 17 47 78 15 C2 0D 01 22 ..-y..Ox.G(...."
0120 25 01 76 43 3A 2E 0F 00 %.vC:...
```

Ordenação

Consideremos uma cadeia de caracteres ASCII alfanuméricos que queremos colocar em ordem crescente.

Utilizaremos um algoritmo elementar de ordenação, baseado em trocas.

O fluxograma apresenta o algoritmo utilizado(pág. 154).

Para exemplificar os passos de execução devemos supor a ordenação de uma cadeia ASCII, inicialmente igual a BCDA.

```
início   — BCDA                          contador = 3
           ↑↑ ponteiro$posterior         trocou = 0
           |
           conteúdo de A

passo 1  — BCDA                          contador = 2
            ↑↑ ponteiro$posterior        trocou = 0
            |
            conteúdo de A
```

```
passo  2  —  BCDA                          contador = 1
               ↑↑ ponteiro$posterior       trocou = 1
               |
              conteúdo de A

passo  3  —  BCAD                          contador = 3
             ↑ ↑ponteiro$posterior         trocou = 0
             |
             conteúdo de A

passo  4  —  BCAD                          contador = 2
              ↑ ↑ponteiro$posterior        trocou = 1
              |
              conteúdo de A

passo  5  —  BACD                          contador = 2
               ↑ ↑ponteiro$posterior       trocou = 1
               |
               conteúdo de A

passo  6  —  BACD                          contador = 3
             ↑ ↑ponteiro$posterior         trocou = 1
             |
             conteúdo de A

passo  7  —  ABCD                          contador = 2
              ↑ ↑ponteiro$posterior        trocou = 1
              |
              conteúdo de A

passo  8  —  ABCD                          contador = 1
               ↑ ↑ponteiro$posterior       trocou = 1
               |
               conteúdo de A

passo  9  —  ABCD                          contador = 3
             ↑ ↑ponteiro$posterior         trocou = 0
             |
             conteúdo de A

passo 10  —  ABCD                          contador = 2
              ↑ ↑ponteiro$posterior        trocou = 0
              |
              conteúdo de A

passo 11  —  ABCD                          contador = 1
               ↑ ↑ponteiro$posterior       trocou = 0
               |
               conteúdo de A
```

Fluxograma: Ordenação

```
                ;**********************************
                ;*ORDENACAO DE UMA CADEIA DE CARACTERES
                ;**********************************
0100                    ORG     100H
                ;*************
                ;*INICIALIZACAO
                ;*************
                DENOVO:
0100 0600               MVI     B,0         ;TROCOU=0
0102 3A2301             LDA     TAMANHO
0105 4F                 MOV     C,A         ;C SERA' O CONTADOR
0106 0D                 DCR     C
0107 212401             LXI     H,CADEIA
                ;
010A 7E         VOLTA:  MOV     A,M
010B 23                 INX     H           ;PONTEIRO$POSTERIOR (HL)
010C BE                 CMP     M           ;SE AS DUAS POSICOES JA' ESTAO
010D FA1A01             JM      NTROCA      ;EM ORDEM NAO TROCA
0110 CA1A01             JZ      NTROCA
                ;***************
                ;*TROCA ELEMENTOS
                ;***************
0113 46                 MOV     B,M
0114 77                 MOV     M,A
0115 2B                 DCX     H
0116 70                 MOV     M,B
0117 23                 INX     H           ;JA' TROQUEI AS DUAS POSICOES
0118 0601               MVI     B,1         ;TROCOU=1
011A 0D         NTROCA: DCR     C           ;DECREMENTA CONTADOR
011B C20A01             JNZ     VOLTA
011E 05                 DCR     B           ;QUANDO TERMINAR DE PERCORRER A CADEIA
011F CA0001             JZ      DENOVO      ;PERGUNTO SE HOUVE TROCA (TROCOU=1 ?)
0122 76                 HLT
                ;***************
                ;*AREAS DE DADOS
                ;***************
0123 04         TAMANHO DB      04
0124 42434441   CADEIA: DB      'BCDA'
0128                    END
```

DUMP antes da execução:

```
0100 06 00 3A 23 01 4F 0D 21 24 01 7E 23 BE FA 1A 01 ..:#.O.!$.~#....
0110 CA 1A 01 46 77 2B 70 23 06 01 0D C2 0A 01 05 CA ...Fw+p#........
0120 00 01 76 04 42 43 44 41 00 ..v.BCDA.
```

DUMP depois da execução:

```
0100 06 00 3A 23 01 4F 0D 21 24 01 7E 23 BE FA 1A 01 ..:#.O.!$.~#....
0110 CA 1A 01 46 77 2B 70 23 06 01 0D C2 0A 01 05 CA ...Fw+p#........
0120 00 01 76 04 41 42 43 44 00 ..v.ABCD.
```

4.4. EXEMPLOS DE PROGRAMAS EM ASSEMBLER Z80

Número de Elementos Negativos em uma Lista

Considere uma lista com oito números, alguns com sinal, outros sem sinal. Determine quantos números da lista são negativos, armazene a quantidade de números negativos em uma posição de memória subseqüente à lista.

lista:	10	33	−6	−40	5	9	−1	12
	0A	21	FA	D8	05	09	FF	0C

Após a execução a posição de memória representada por RESULT, contém 3 (ver programa na pág. 157).

Conversão Hexadecimal para ASCII

Considere um número armazenado em uma posição de memória (8 bits), que pode ser representado por dois dígitos hexadecimais. Converta este número em dois caracteres ASCII, que corresponda a representação hexadecimal, e armazene em duas posições de memória.

ANTES	*DEPOIS*
NUM : B4	NUM : B4
	ALTA : 42 = "B"
	BAIXA : 34 = "4"

(Ver programa na pág. 158).

```
                              .Z80
                              ;***************************************
                              ;*
                              ;*NUMERO DE ELEMENTOS NEGATIVO DE UMA LISTA
                              ;*
                              ;***************************************
                              ;***********
                              ;*INICIALIZA
                              ;***********
0000'   3E 00                 INIT:   LD      A,0      ;ZERO CONTADOR DE NUMEROS NEGATIVO
0002'   21 0013'                      LD      HL,NUM
0005'   46                            LD      B,(HL)   ;B SERA' MEU CONTADOR
0006'                         VOLTA:
0006'   23                            INC     HL
0007'   CB 7E                         BIT     7,(HL)   ;TESTA SE NUMERO E' NEGATIVO
0009'   28 01                         JR      Z,POS    ;SE NAO FOR,NAO INCREMENTA CONTADOR
000B'   3C                            INC     A        ;INCREMENTA NUMERO DE NEGATIVOS
000C'                         POS:
000C'   05                            DEC     B        ;DECREMENTA CONTADOR
000D'   20 F7                         JR      NZ,VOLTA
000F'   32 001C'                      LD      (RESULT),A
0012'   76                            HALT
0013'   08                    NUM:    DB      8        ;NUMERO DE ELEMENTOS DA LISTA
0014'   0A 21 FA D8           LISTA:  DB      10,33,-6,-40,5,9,-1,12
0018'   05 09 FF 0C
001C'   00                    RESULT: DB      0        ;GUARDA O NUMERO DE NEGATIVOS
                                      END INIT
```

```
                        .Z80
                        ;************************
                        ;*
                        ;*CONVERSAO DE HEXA-)ASCII
                        ;*
                        ;************************
0000'   3A 0025'        INITIO: LD      A,(NUM)
0003'   E6 0F                   AND     0FH     ;ACC FICA COM OS 4 BITS MENOS SIGNIF.
0005'   CD 001D'                CALL    ASCII
0008'   32 0027'                LD      (BAIXA),A
                        ;
000B'   3A 0025'                LD      A,(NUM) ;RECARREGO PARA PEGAR PARTE BAIXA
000E'   CB 3F                   SRL     A       ;
0010'   CB 3F                   SRL     A       ; DESLOCA OS 4 BITS MAIS SIGNIFICATIVOS
0012'   CB 3F                   SRL     A       ; PARA POSICAO MAIS 'A DIREITA DO ACC.
0014'   CB 3F                   SRL     A       ;
0016'   CD 001D'                CALL    ASCII
0019'   32 0026'                LD      (ALTA),A
001C'   76                      HALT
                        ;***********************************
                        ;*ROTINA QUE PASSA CONTEUDO DO ACC--)ASCII
                        ;***********************************
001D'                   ASCII:
001D'   C6 30                   ADD     A,30H   ; TRANSFORMO EM NUMERO
001F'   FE 3A                   CP      3AH
0021'   D8                      RET     C       ; RETORNA SE FOR NUMERO ('0'--)'9')
0022'   C6 07                   ADD     A,7     ; SENAO RETORNAR TRANSFORMA EM LETRA ('A'--)'F')
0024'   C9                      RET
                        ;*************
                        ;*AREA DE DADOS
                        ;*************
0025'   B4              NUM:    DB      0B4H
0026'   00              ALTA:   DB      0
0027'   00              BAIXA:  DB      0

                                END     INITIO
```

```
                          .Z80
                          ;****************
                          ;*
                          ;*BUSCA E INSERCAO
                          ;*
                          ;****************
0000'   3A 002E'          INITIO: LD      A,(CARAC); PEGO O CARACTER
                          ;
0003'   ED 4B 0039'               LD      BC,(TAM); BC SERA' O CONTADOR
0007'   21 003B'                  LD      HL,LISTA ; APONTA PARA 1. ELEMENTO DA LISTA
000A'   ED B1                     CPIR             ; VEJO SE CARACTERE JA' ESTA' NA LISTA
000C'   ED 4B 0039'               LD      BC,(TAM) ; RETORNO O CONTADOR
0010'   28 1B                     JR      Z,EXISTE
                          ;
0012'   21 003B'                  LD      HL,LISTA ; APONTO NOVAMENTE P/ 1. ELEMENTO
0015'   ED A1             BUSCA:  CPI              ; E COMPARO
0017'   FA 001F'                  JP      M,INSERE
001A'   20 F9                     JR      NZ,BUSCA
                          ;****************************************
                          ;*COLOCO CARACTERE NA ULTIMA POSICAO DA LISTA
                          ;****************************************
001C'   77                        LD      (HL),A
001D'   18 0E                     JR      FIM
                          ;***********************
                          ;*INSIRO NO MEIO DA LISTA
                          ;***********************
001F'   03                INSERE: INC     BC
0020'   C5                        PUSH    BC       ; SALVO CONTEUDO ATUAL DO CONTADOR
0021'   2B                        DEC     HL
0022'   11 002F'                  LD      DE,AUX
0025'   ED B0                     LDIR             ; TRANSPORTO A LISTA PARA UMA POS. AUXILIAR
0027'   C1                        POP     BC       ; RETORNO VALOR DO CONTADOR
0028'   1B                        DEC     DE       ; APONTO PARA ULTIMO ELEMENTO TRANSPORTADO
0029'   EB                        EX      DE,HL
002A'   ED B8                     LDDR             ; TRAGO DE VOLTA
002C'   12                        LD      (DE),A   ; E INSIRO NOVO ELEMENTO
                          ;*************************
                          ;*TERMINO O PROGRAMA SENDO:
                          ;*************************
002D'                     EXISTE:          ; CARACTER JA' ESTA NA LISTA
002D'                     FIM:             ; CARACTER FOI INSERIDO NA LISTA
002D'   76                        HALT
                          ;*************
                          ;*AREA DE DADOS
                          ;*************
002E'   43                CARAC:  DB      'C'
002F'                     AUX:    DS      10
0039'   000A              TAM:    DW      10
003B'   41 42 44 48       LISTA:  DB      'A','B','D','H','I','J','L','X','Y','Z'
003F'   49 4A 4C 58
0043'   59 5A
0045'                             DS      2        ; PREVISAO PARA INSERCAO
                                  END     INITIO
```

Busca e Inserção

Considere uma lista de caracteres, ASCII, ordenada em ordem crescente. Dado um novo caracter, caso este caracter não exista na lista, insira-o de modo que a lista permaneça ordenada.

Lista: "A" "B" "D" "H" "I" "J" "L" "X" "Y" "Z"

Após a inserção do caracter C.

Lista: "A" "B" "**C**" "D" "H" "I" "J" "L" "X" "Y" "Z"

A utilização dos recursos adicionais disponíveis no microprocessador Z80, facilita bastante a programação, em comparação com as limitações dos Micros 8080A/8085, porém é importante considerar que os recursos disponíveis nos Micros 8080A/8085 são suficientes para o desenvolvimento da maioria das aplicações envolvendo microprocessadores; portanto é muito comum a utilização em Micros Z80 apenas do subconjunto correspondente ao 8080A/8085, garantindo uma maior portabilidade das aplicações (ver programa na pág. 159).

4.5. EXERCÍCIOS

1. Escreva um programa em Assembler 8080/8085, que calcule a soma de uma série de números de 16 bits. Assuma que a soma possa ser armazenada em 16 bits.

2. Escreva um programa em Assembler 8080/8085, que determine o número de bits 1 existente em uma cadeia binária. A cadeia está organizada da seguinte forma: 1 byte (tamanho da cadeia), 1 ou mais bytes (cadeia propriamente dita).

3. Escreva um programa em Assembler Z80, que localize uma cadeia de caracteres ASCII, em uma outra de tamanho igual ou superior. O programa deverá indicar se a cadeia foi ou não encontrada, e em caso afirmativo, informar a sua localização.

4. Escreva em Assembler 8080/8085, um subprograma para converter números binários para BCD. O subprograma recebe no par DE, o apontador para um par de endereços: endereço de memória do número binário a ser convertido; endereço de memória do número em BCD.

Exemplo: DE $= 120_{16}$ $(0130_{16}) = 1A$
$(120_{16}) = 01_{16}$
$(121_{16}) = 30_{16}$
$(122_{16}) = 01_{16}$
$(123_{16}) = 31_{16}$

Resultado: $(0131_{16}) = 02$
$(0132_{16}) = 06$

5. Repita o exercício anterior para BCD compactado, isto é, cada byte armazena dois dígitos decimais. No exemplo do exercício 4 teremos $(0131_{16}) = 26$.

6. Escreva um programa em Assembler 8080/8085, que faça a divisão de um número de 16 bits, sem sinal, por um número de 8 bits sem sinal. O programa deve fornecer como resultado o quociente e o resto da divisão.

7. Escreva um programa em Assembler Z80, que calcule o dígito verificador, de uma cadeia de números em BCD compactado. Utilize o seguinte dígito verificador: 1, 3, 5 Mod. 10.

Exemplo:

Considere o seguinte número: 637243

$(0130_{16}) = 03$ (tamanho da cadeia em bytes)
$(0131_{16}) = 63$
$(0132_{16}) = 27$
$(0133_{16}) = 48$
DÍGITO VERIFICADOR = $6 + \underline{3} \times 3 + \underline{5} \times 2 + 7 + \underline{3} \times 4 + \underline{5} \times 8 = 84$ MOD 10
DÍGITO VERIFICADOR = 4

8. Considerando o exemplo de Busca e Inserção, altere o programa escrito em Assembler Z80, para admitir também a remoção de um caracter, em uma lista ordenada em ordem crescente.

Exemplo:

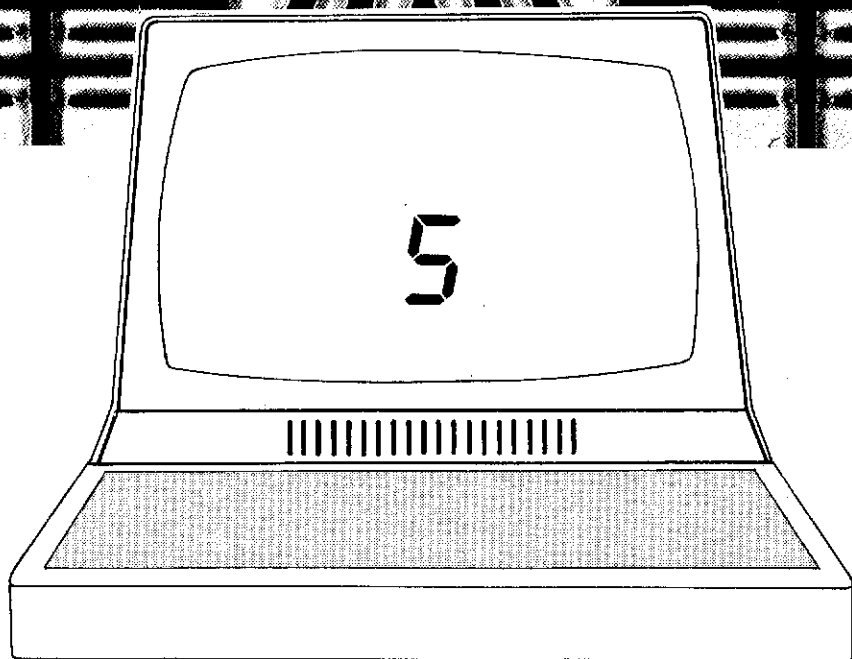
Lista: "A" "B" "C" "D" "H" "J" "L" "X" "Y" "Z"

Remover "J"
Lista: "A" "B" "C" "D" "H" "L" "X" "Y" "Z"

9. Escreva um subprograma em Assembler 8080/8085, que simule numa calculadora de 4 operações (soma, subtração, multiplicação e divisão). A calculadora admitirá operações com operandos de 6 dígitos BCD. O subprograma recebe os seguintes parâmetros de entrada:

Registro C = 0 — soma
1 — subtração
2 — multiplicação
3 — divisão

Par de registro DE: aponta para uma área de 6 bytes que contém os operandos. Os primeiros três bytes contêm o primeiro operando, e os três bytes seguintes o segundo.

O subprograma fornecerá o resultado na área reservada ao primeiro operando, isto é, o par de registros DE retornará apontando para o resultado.

PROJETO COM MICROPROCESSADORES

5.1. SISTEMAS COM MICROPROCESSADORES

Como visto no Cap. 2, o processador é composto de um conjunto de registros de depósito, circuitos aritméticos e circuitos de controle. No passado cada um desses elementos era construído manualmente a partir de válvulas (posteriormente transistores), fios, resistências etc. Na 2ª geração de computadores, onde os transistores foram intensamente utilizados, o processo de construção continuou o mesmo.

Com o aparecimento do circuito integrado, uma quantidade muito maior de circuitos pôde ser colocada dentro de um mesmo componente reduzindo não só o tamanho mas também o tempo de montagem, já que, para montar um registro de dados, era necessário colocar e soldar vários transistores ao passo que um único integrado pode conter um registro completamente. Este foi o ponto base dos computadores de 3ª geração.

A evolução da tecnologia eletrônica não parou aí e nem há indícios de que vá parar tão cedo. No início da década, já havia condição de colocar em um único componente toda a unidade de processamento de um computador incluindo aritmética, controle e diversos registradores. Eram os microprocessadores que surgiam.

Apesar do constante e veloz progresso que já permite colocar cerca de 10 vezes mais circuitos num elemento do que nos primeiros microprocessadores, as características lançadas pelas primeiras famílias de microprocessadores permanecem válidas e populares ainda hoje e, possivelmente, sobreviverão um bom tempo apesar dos espetaculares produtos recentemente lançados.

Tipos de Processadores

Existem vários microprocessadores no mercado; podemos classificá-los em microprocessadores de uma pastilha, microprocessadores sem ROM, microprocessadores sem memória e microprocessadores em fatias.

Microprocessadores de uma pastilha. São sistemas que contêm numa mesma embalagem (pastilha) processador, memória de programas, memória de dados e linhas de entrada/saída (Fig. 5.1).

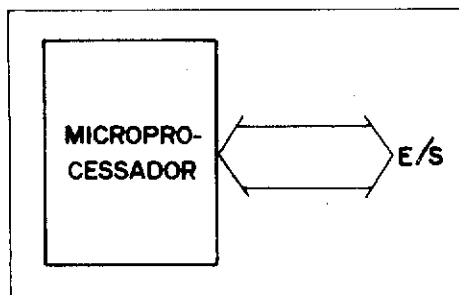



Fig. 5.1 — Microprocessador de uma pastilha.

Como tudo é feito em uma só pastilha só há vias internas (endereço, dados, controle); basicamente, os pinos estão disponíveis para linhas de entrada/saída.

À primeira vista podem parecer os micros ideais já que tudo está concentrado e simplificado. Ocorre, no entanto, que alguns pontos são sacrificados nesse tipo de sistema:

1. O processador é mais simples, normalmente com menor variedade de instruções e menor velocidade.
2. Pouca memória disponível, no máximo 2K bytes de ROM e 128 bytes (*não falta o "K" não, é 128 mesmo!*) de RAM, não havendo possibilidade de expansão já que não existe via de endereço nem de dados.
3. A memória ROM normalmente oferecida só pode ser gravada em fábrica o que inviabiliza o uso do sistema em pequenas quantidades.

Este tipo de microprocessador, portanto, é adequado para uso em bens de consumo (brinquedos, máquinas de costura, fornos etc.) porque é extremamente barato quando usado em grandes quantidades e também porque a programação, sendo simples, pode se acomodar nos 2K disponíveis.

Nesta categoria se enquadram, dentre outros:

Z8 (zilog), 8021, 8048 (intel)

Microprocessadores sem ROM. Tentando contonar o grande inconveniente de só ser viável em grandes quantidades, foram lançados microprocessadores semelhantes aos de uma pastilha porém sem memória de programa (ROM) interna.

Neste tipo, já há necessidade de vias de endereços, dados e controle para comunicação com a ROM apesar da memória RAM interna.

Como exemplo dessa categoria podemos citar o processador 8025 (Intel).

Microprocessadores sem memória. O tipo mais popular, no entanto, é dos processadores cujas memórias (ROM e RAM) não estão contidas na mesma embalagem do processador. Todos os demais componentes se comunicam com o processador através das vias de endereços, dados e controle (Fig. 5.2).

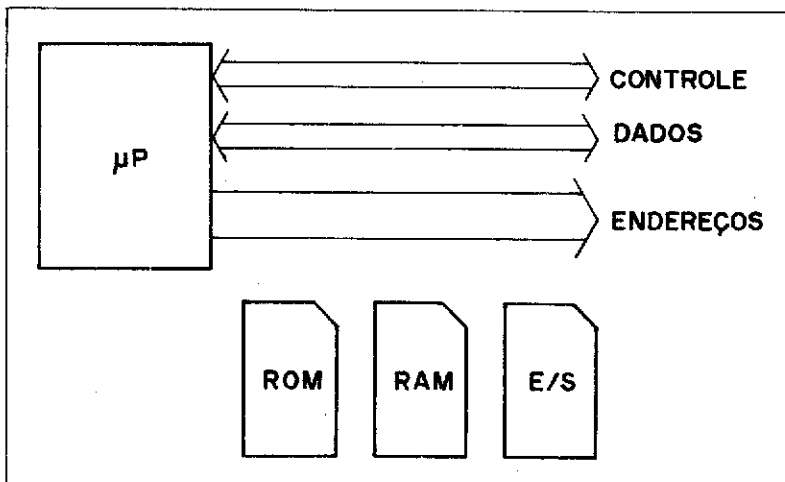



Fig. 5.2 — Microprocessador sem memória.

Em geral, a embalagem do microprocessador contém um processador relativamente complexo, incluindo um número razoável de registros de depósito para trabalho. Um critério que tem sido utilizado para classificar os microprocessadores é quanto ao número de bits transferidos simultaneamente pela via de dados. Existem processadores de 4, 8, 16 e 32 bits.

Esta classificação, apesar de um tanto falha, tem sido conveniente porque pode-se observar uma coerência nas demais características mesmo considerando-se diferentes fabricantes.

Os microprocessadores de 4 bits foram os primeiros a aparecer (INTEL 4004) e modelos mais aperfeiçoados (p. ex. INTEL 4040) continuam desempenhando papel relevante como controladores para aplicações relativamente simples.

Os processadores de 8 bits surgiram pouco depois e, após um período de indefinição quanto ao mercado, tornaram-se extremamente populares (8008, 8080, 8085, INTEL — Z80, Zilog — 6800 Motorola etc.) tendo sido os produtos-base viabilizando os computadores pessoais. Os micros de 8 bits não estão fadados a desaparecer meramente porque micros de 16 e 32 bits estejam sendo lançados no mercado. Tem-se observado um grande número de micros de 8 bits que continuam sendo lançados mesmo por fabricantes que já lançaram micros de 16 bits. O ponto é que, para diversas aplicações, não há o que ganhar usando processadores mais potentes; pelo contrário, pode-se perder em tempo de projeto, custo e espaço.

A Família 8080. Dentre os micros de 8 bits deve-se destacar o papel dos integrados da família 8080. O primeiro micro desta família, o INTEL 8080 foi lançado em 1974 para substituir o 8008; seu sucesso foi tão grande, que logo foram lançados micros equivalentes, isto é, com mesma arquitetura e linguagem de máquina, por diversos fabricantes. Posteriormente surgiu o Z80 mantendo a mesma linguagem de máquina, com algumas extensões, de modo a usar o mesmo *software* já existente. Em 1978 a Intel lançou o 8085 destinado a substituir o 8080 porém mantendo o mesmo conjunto de instruções.

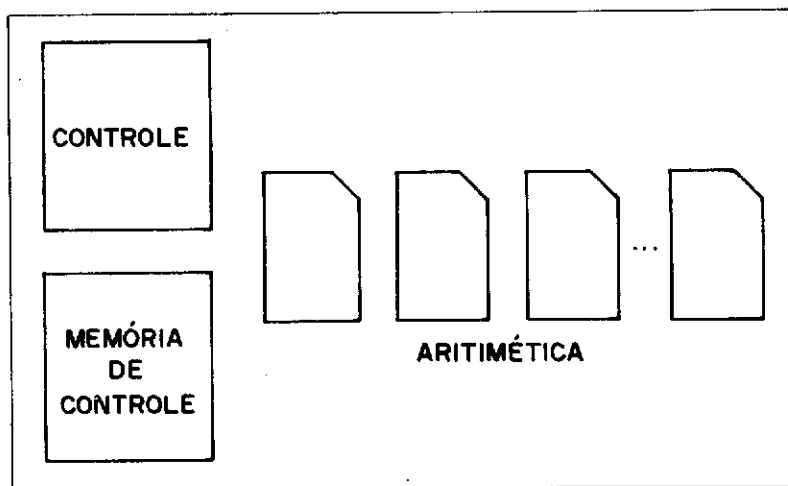



Fig. 5.3 — Microprocessador "bit-slice"

Observa-se, de fato, uma grande padronização dos microcomputadores em torno da família 8080 o que permitiu o aparecimento de um volume espetacular de *software* a baixo custo.

Existem, no entanto, outros micros muito bem sucedidos no mercado; porém a família 8080 continua exercendo um papel de destaque dentre os micros de 8 bits. Os dois membros mais "nobres" desta família são o 8085 e o Z80 aos quais destinaremos mais estudo.

Microprocessadores em fatias — "BIT-SLICE". Um outro tipo de processador usual é o chamado processador "bit-slice". Na verdade o processador consiste de um conjunto de pastilhas incluindo "fatias de unidades aritméticas". Isto porque cada fatia é uma unidade aritmética de 2 bits e várias fatias devem ser ligadas para se obter o tamanho desejado (Fig. 5.3).

A nomenclatura é um tanto imprópria já que o processador está espalhado dentre várias pastilhas. Normalmente esses processadores usam uma tecnologia chamada bipolar que resulta em maior velocidade de operação, maior consumo de energia e maior espaço ocupado (devido a menor densidade).

5.2. SISTEMA INTEL 8085 — MÍNIMO

Nesta seção explicaremos o uso dos componentes do Sistema Intel MCS-85 e sua configuração mínima. Entenderemos por configuração mínima um sistema microcomputador usando 3 pastilhas:

— O microprocessador (8085)
— Memória ROM e E/S (8355 ou 8755)
— Memória RAM, E/S e relógio (8155 ou 8156)

Estes três componentes nos permitirão montar um sistema microcomputador completo com 2K bytes de ROM, 256 bytes de RAM e relógio (além de várias linhas de entrada/saída).

Microprocessador 8085A

Na Fig. 5.4 vemos o diagrama de pinagem e um diagrama funcional típico. Os pinos 1 e 2 (X_1, X_2) estarão sempre ligados a um cristal (para obter a freqüência de operação) e os pinos 40 e 20 (V_{cc} e V_{ss}) estarão sempre ligados respectivamente a +5V e à terra; por esta razão, estes pinos serão sempre omitidos dos diagramas funcionais.

Vias de endereços e dados. O 8085 usa 16 linhas de endereçamento, podendo, conseqüentemente, endereçar 64K bytes de memória. No entanto, com o objetivo de reduzir o número de pinos na embalagem, 8 dessas 16 linhas são multiplexadas com os dados; isto é, quando o 8085 faz uma referência à memória, ele envia os bits A_0 — A_7 (bits menos significativos do endereço) pelos pinos

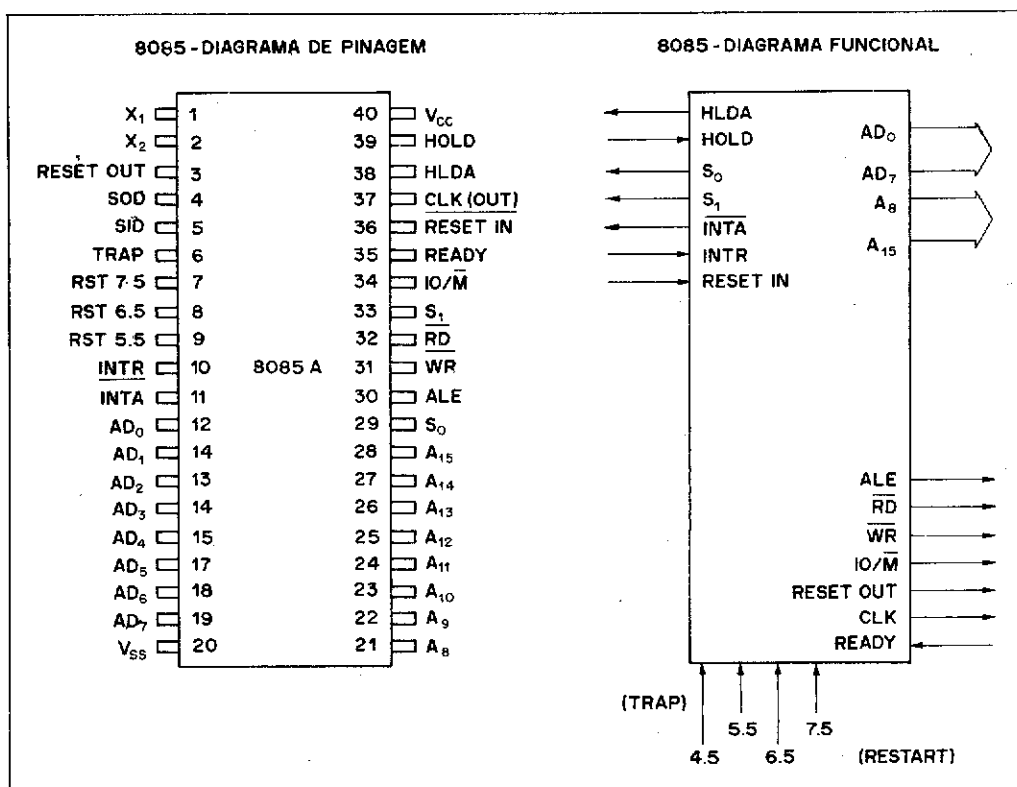



Fig. 5.4 – Microprocessador 8085 A.

12 a 19 (a via de dados) e, mais tarde, usará estes mesmos pinos (12 a 19) para mandar (no caso de uma escrita) ou receber (no caso de uma leitura) 8 bits de dados.

Dessa forma, o endereço de memória completo só aparece durante pouco tempo.

Como é que a memória sabe se o processador está mandando endereço ou dados nestes pinos? Para isso existe um sinal de controle no pino 30 (ALE — "Address Latch Enable") que indica se estas linhas contêm endereço (ALE = 1) ou dados (ALE = 0) (Fig. 5.5).

Na verdade a memória terá que "lembrar" o conteúdo desses 8 bits de endereço durante todo o tempo de acesso; isto será explicado adiante.

Endereços de entrada/saída. Quando o 8085 faz uma referência à memória (leitura ou escrita sendo $IO/\overline{M} = 0$), ele coloca os 16 bits de endereço divididos em duas partes como já foi dito. No entanto, se o 8085 faz uma referência a uma porta de entrada/saída ($IO/\overline{M} = 1$) não há necessidade de usar 16 bits já que os endereços de E/S só requerem 8 bits (0 a 255).

Como o 8085 usa as linhas neste caso?

O que é feito é que o endereço (de 8 bits) é repetido tanto nas posições A_{15} — A_8 como nas posições A_7 — A_0.

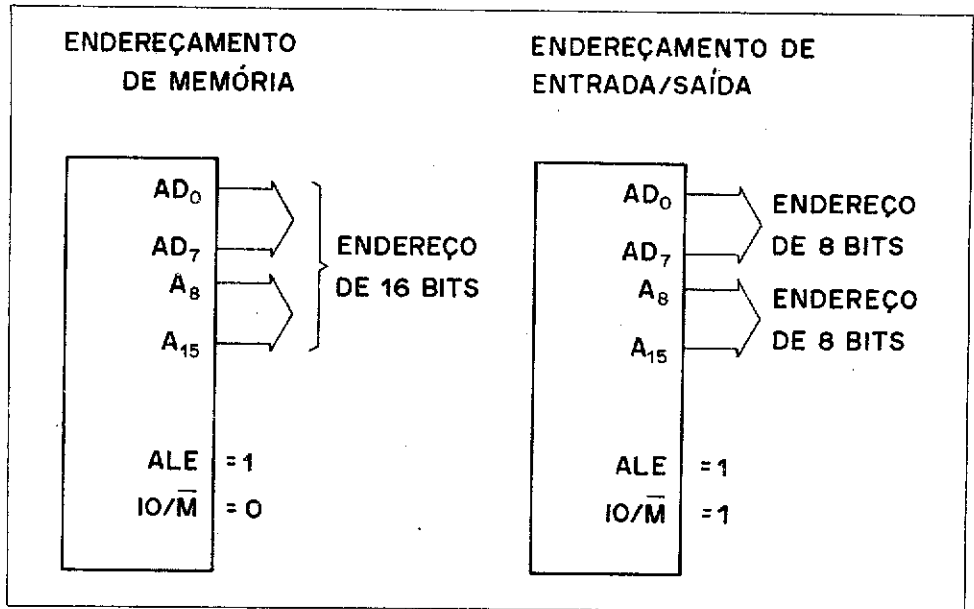



Fig. 5.5 – Endereços de Memória e de Entrada/Saída.

Na Fig. 5.5 ilustramos a situação durante o 1º estado de máquina quando os endereços são enviados (ALE = 1); nos estados seguintes (ALE = 0) a via de dados será usada para transferir dados tanto para referências à memória como para entrada/saída.

Via de controle. Além das vias de endereços e dados, existem diversos sinais de controle necessários ao funcionamento do sistema. Um deles já foi visto (ALE); estudaremos agora alguns outros: CLK, $\overline{RD}$, $\overline{WR}$, IO/$\overline{M}$, READY.

CLK — é o relógio do processador. Neste pino o processador envia um sinal que pode ser usado por outros componentes como base de tempo.

$\overline{RD}$ — é um sinal (invertido) de leitura; isto é, se for zero indica que o 8085 deseja uma leitura.

$\overline{WR}$ — é um sinal (invertido) de escrita; isto é, se for zero indica que o 8085 deseja uma escrita.

IO/$\overline{M}$ — indica se o 8085 está executando uma operação de entrada/saída (IO/$\overline{M}$ = 1) ou não (IO/$\overline{M}$ = 0). Isto *normalmente* indica se a referência é para uma porta ou para uma posição de memória.

READY — é um sinal que *entra* no processador normalmente para sincronizar operação com memórias lentas. Se for zero, o 8085 pára de funcionar até que este sinal volte a ser 1.

Memória ROM e Entrada/Saída (8355 ou 8755)

Outro elemento importante para compor um sistema MCS85 é a pastilha de memória ROM e entrada/saída que existe em duas versões: 8355 com memória ROM gravada em fábrica (por máscara) e 8755 com memória EPROM programável pelo usuário e apagável com luz ultravioleta. Exceto pelo modo de gravar as informações, as duas pastilhas são funcionalmente idênticas e têm 2K bytes de memória e duas portas de entrada/saída de 8 bits cada uma.

Os diagramas de pinagem e funcional encontram-se na Fig. 5.6.

Como existem 2K bytes de memória, são necessárias 11 linhas de endereço: AD_0 a AD_7 e A_8 a A_{10}. Tendo em vista que estas pastilhas foram desenvolvidas especificamente para serem usadas com o micro 8085 que mistura via de endereços com via de dados, os 8 bits menos significativos do endereço são multiplexados com os dados da mesma forma que no 8085. Usando o sinal de controle ALE dado pelo processador, estas pastilhas não só reconhecem *quando* a via de dados contém bits de endereço, mas também "recordarão" a configuração dêstes bits A_0 — A_7 até o próximo ciclo.

Além dos 2K bytes de memória existem quatro registros "depósito" (o diagrama funcional só mostra dois). Os dois que aparecem (A e B) são portas de entrada/saída cujos bits estão ligados ao mundo exterior; os outros dois, que são chamados registros de direção de dados A e B, são necessários para que possamos especificar cada bit da porta correspondente como sendo de entrada ou saída.

Assim, o registro de direção de dados A (DDR A) determina como os bits na porta A serão usados; um bit 1 no DDR indica que o bit correspondente na porta será uma saída, um bit zero indica uma entrada. O mesmo raciocínio aplica-se ao DDR B e à porta B.

Exemplo: Se o DDR A contém 11110000, então os bits 0 a 3 da porta A serão entradas e os bits de 4 a 7 serão saídas.

Observação: Normalmente, assim que um sistema é ligado, o estado inicial de vários registros é posto em zero. Os DDR, dessa forma, são zerados passando a indicar que todos os bits das portas A e B são de entrada até que, mais tarde, o processador explique qual a verdadeira configuração desejada.

Este procedimento é, na verdade, o mais sensato porque configurando todos os bits como entrada, eles não interferirão com o exterior. Se, erradamente, um bit fosse configurado para saída e houvesse um sinal externo, poderia haver conflito resultando em dano no circuito.

Uso das Portas de E/S

Como se pode ler ou escrever nestes registros DDR e portas?

Cada registro tem um endereço de dois bits, tomados dos bits 0 — 1 enviados através da via de endereços como se segue:

00 — Porta A
01 — Porta B
10 — DDR A
11 — DDR B

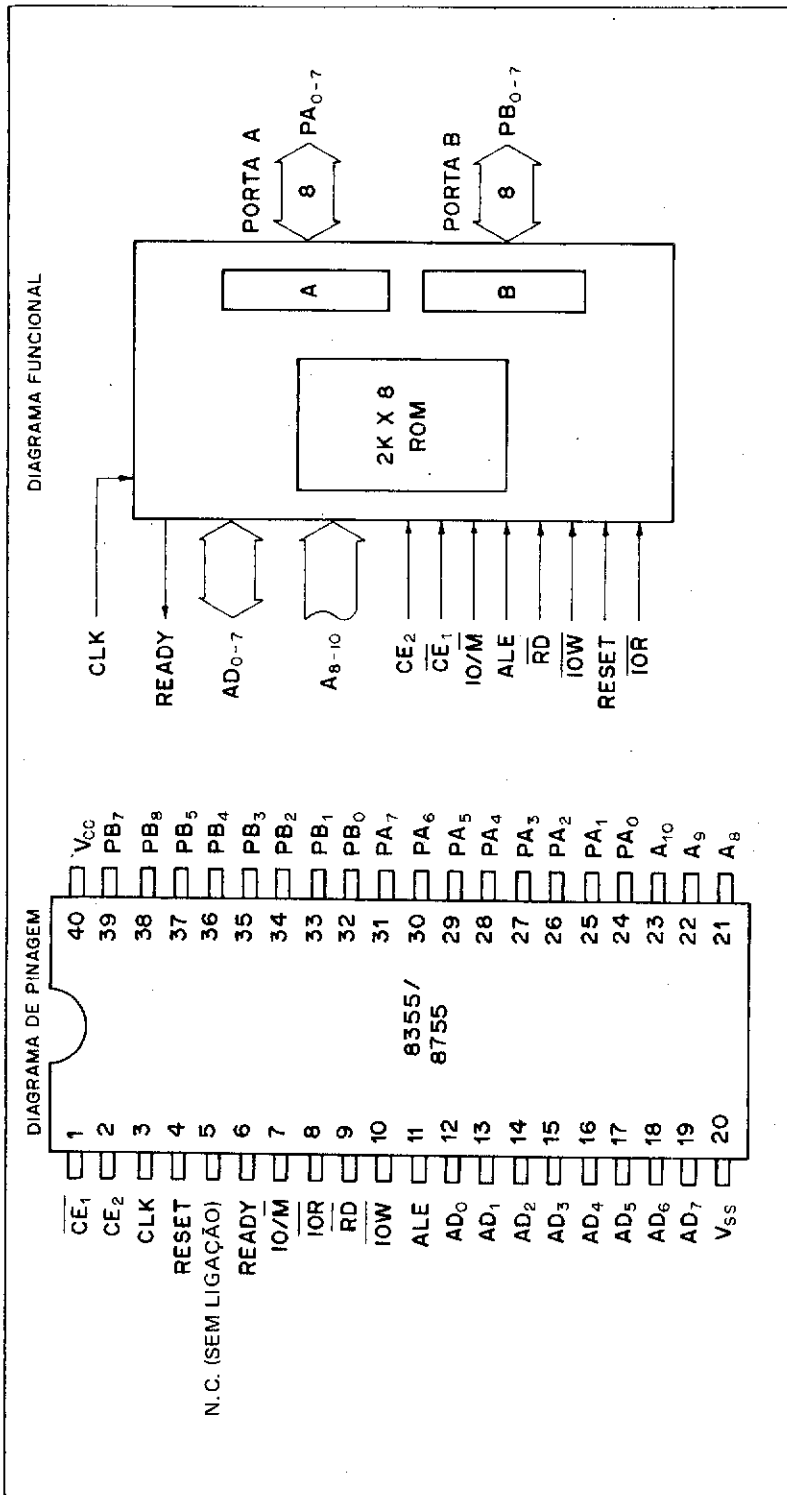



Fig. 5.6 — Circuito ROM 8355/8755.

Neste ponto o leitor poderá levantar uma dúvida crucial! Já que também existe memória na mesma pastilha, cujo endereço vai também pela mesma via, como adivinhar se o processador se refere a uma posição de memória ou a um desses registros? Exemplificando: como saber se um endereço 0000 se refere à posição de memória 0000 ou à porta A (que também tem o endereço 0000)?

Para evitar essa confusão o processador deverá sempre informar se está fazendo referência a uma posição de memória ou a uma entrada/saída. Isto é feito através do sinal de controle $IO/\overline{M}$. Se for 1 (IO) então deve-se considerar que o endereço se refere a um registro; se for 0 ($\overline{M}$) o endereço se refere à memória.

O processador 8085 faz $IO/\overline{M} = 1$ sempre que está executando uma instrução de E/S (IN ou OUT). Se o sinal $IO/\overline{M}$ da pastilha ROM estiver ligado ao sinal $IO/\overline{M}$ que sai do processador (o que é mais provável) então as instruções IN e OUT serão referentes aos registros ao passo que as demais instruções referir-se-ão à memória. Existe uma técnica, denominada "E/S mapeada na memória" (Memory Mapped I/O) que procederá de forma diferente.

Deve-se deixar bem claro que os registros de direção (DDR) devem ser carregados com a configuração de bits desejada *antes* que as portas sejam usadas; isto é, o processador deve executar um programa que inicializa os DDR's. A direção de cada linha pode ser mudada a qualquer momento se isto for desejado.

Outros sinais de controle

IOR, $\overline{CE}_1$, CE_2, $\overline{RD}$, $\overline{IOW}$, CLK

IOR — O sinal IOR será negligenciado em nosso estudo e permanentemente ligado em +5V.

$\overline{CE}_1$, CE_2 — Os sinais CE_1 e CE_2 são sinais de habilitação; a *pastilha estará em funcionamento somente quando $\overline{CE}_1$ for zero e CE_2 for 1.*

Todas as pastilhas costumam ter pelo menos uma habilitação. Como várias pastilhas são normalmente ligadas a uma mesma via, o sinal de habilitação é usado para ligar apenas aquela que desejarmos.

Os sinais de habilitação sempre são gerados a partir da linha de endereços seja usando diretamente uma linha de endereço (seleção linear) ou através de um decodificador.

$\overline{RD}$ — É um sinal invertido de leitura. Se a pastilha estiver habilitada ($\overline{CE}_1 = 0$, $CE_2 = 1$) será feita uma leitura seja da ROM ou de um dos registros (conforme seja especificado por $IO/\overline{M}$). Este sinal deve ser ligado ao $\overline{RD}$ do 8085.

$\overline{IOW}$ — É um sinal invertido de escrita. No entanto, como a memória existente não pode ser escrita, já que é uma ROM, a escrita só faz sentido para um dos registros. Assim, a despeito do que seja indicado por $IO/\overline{M}$, se este sinal for zero o registro indicado pelos 2 bits menos significativos de endereço será gravado com o conteúdo das linhas de dados. Portanto, se $\overline{IOW}$ for ligado ao $\overline{WR}$ do 8085, qualquer operação de escrita que ocorrer enquanto a pas-

tilha estiver habilitada, afetará um dos registros. Por exemplo, se a pastilha estiver ligada para os endereços 0000 — 1FFF, uma escrita na posição de memória 0372 na verdade causará uma escrita no registro 2, o DDR da porta A.

CLK
READY
$\overline{RESET}$ Estes sinais devem ser ligados aos sinais respectivos adivindos do 8085.

Memória RAM, Entrada/Saída e Relógio (8155 ou 8156)

Estas pastilhas dispõem de 256 bytes de RAM, três portas de entrada/saída, sendo duas de 8 bits e uma de 6 bits e um relógio. Os diagramas de pinagem e funcional são mostrados na Fig. 1.7.

A diferença entre a 8155 e a 8156 está somente na entrada de habilitação: na 8155 a habilitação $\overline{CE}$ é invertida, ou seja, a pastilha funciona quando $\overline{CE} = 0$ ao passo que na 8156 a habilitação é positiva e a pastilha funciona quando $CE = 1$.

À primeira vista pode parecer estranho ter tantas funções diferentes em uma mesma pastilha tornando sua utilização mais complicada. O leitor deve, no entanto, ter em mente que, justamente devido a sua múltipla utilidade, uma destas pastilhas evita a necessidade de se lançar mão de toda uma variedade de outros circuitos.

Do mesmo modo que a ROM (8755-8355) estas pastilhas também retratam os 8 bits menos significativos do endereço que são misturados na via de dados.

A utilização da memória dentro destas pastilhas é elementar. Se IO/$\overline{M}$ for zero, a pastilha subentende que a parte referente à memória deve ser utilizada e não afetará as entradas/saídas nem o relógio. Como apenas 256 bytes estão disponíveis, 8 bits são suficientes para endereçamento e são obtidos dos pinos de endereçamento/dados AD_7 — AD_0. Note-se que estes 8 bits são "fotografados" para que os mesmos pinos possam ser usados para transferir dados posteriormente.

AD_7 a AD_0 devem ser ligados à via de dados e ALE deve ser ligado ao sinal ALE do processador.

Uso de entrada/saída e relógio. Dentro das 8155/8156 existem, além da memória, sete registros-depósito onde são armazenadas informações funcionais. Já que existem três portas, o leitor pode já ter adivinhado que três desses registros são, na verdade, portas ligadas ao exterior. As portas A e B são de 8 bits e a porta C é de 6 bits.

Ao contrário do que ocorre na 8355/8755, não se pode especificar individualmente cada linha para ser entrada ou saída; ao invés disso, pode-se programar *todas* as linhas da porta A para serem entrada *ou* saída, todas as linhas da porta B para serem entrada *ou* saída e, finalmente, uma dentre as quatro possíveis alternativas para a porta C:

1. (entrada) todos os 6 bits são entrada;
2. (saída) todos os 6 bits são saída;
3. (controle/saída) bits 0 a 2 são controles da porta A; bits 3 a 5 são saída.

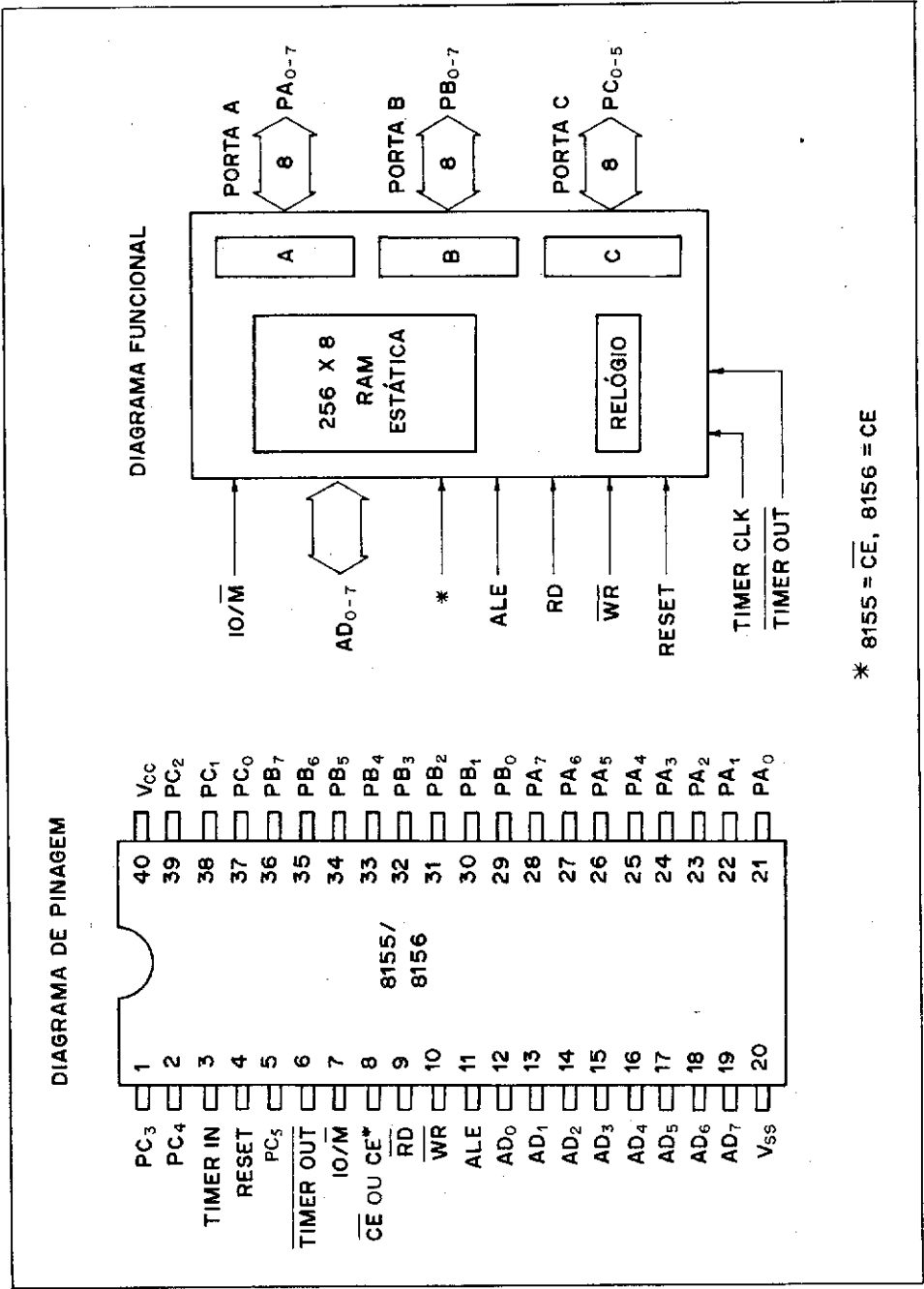



Fig. 5.7 — Circuito RAM, E/S, relógio 8155/8156.

4. (controle) bits 0 a 2 são controles da porta A;
bits 3 a 5 são controles da porta B.

As alternativas 1 e 2 são muito simples; a operação das portas será semelhante à das portas da 8755. Quando a porta C for configurada como controle, a operação da porta a ser controlada (A ou B) será substancialmente distinta como veremos adiante.

Entrada/saída com controle. Existem três sinais de controle que poderão funcionar usando a porta C:

INTR — (Interrupção) — indica que a porta precisa de atenção.

BF — ("Buffer Full" — Registro cheio) — indica a existência de dados na porta.

$\overline{STB}$ ("Strobe" — validação) — indica que um dado recebido na porta deve ser "fotografado" para entrar no computador (entrada) ou então que um dado presente na porta já foi "fotografado" por algum dispositivo externo (saída).

Observe-se que a porta C sendo configurada para controlar uma das outras (seja A, seja A e B) as portas controladas se comportarão diferentemente. Uma entrada/saída controlada é dita trabalhar no modo "validação" a que também poderíamos chamar modo "protocolado" em oposição ao chamado modo básico (quando não há controle).

No modo protocolado uma porta de entrada "fotografa" os dados no momento que recebe o sinal de validação $\overline{STB}$ (no modo básico o dado não é fotografado). Por outro lado, deve ficar claramente entendido que uma porta controlada não funcionará sem o sinal $\overline{STB}$, pois sem ele o dado não será fotografado e, conseqüentemente, jamais poderá ser lido!

Numa porta configurada para entrada, a validação fará com que o registro correspondente (A ou B) guarde um "retrato" do estado das linhas de entrada neste momento e fará também com que seja ligado o sinal BF — registro cheio. Assim que termina o sinal de validação o sinal INTR (interrupção) será ligado, por um circuito interno à pastilha, com a finalidade de pedir um atendimento

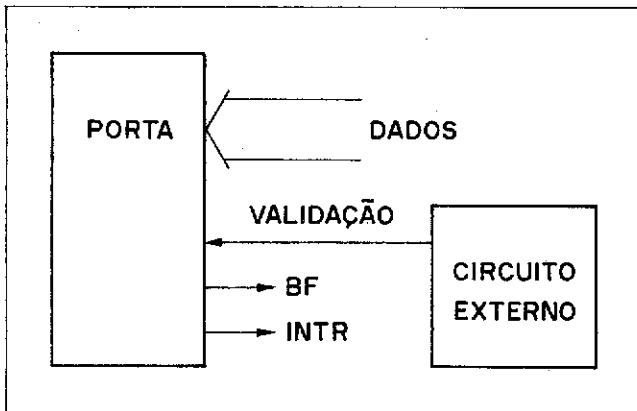



Fig. 5.8 Entrada/Controlada

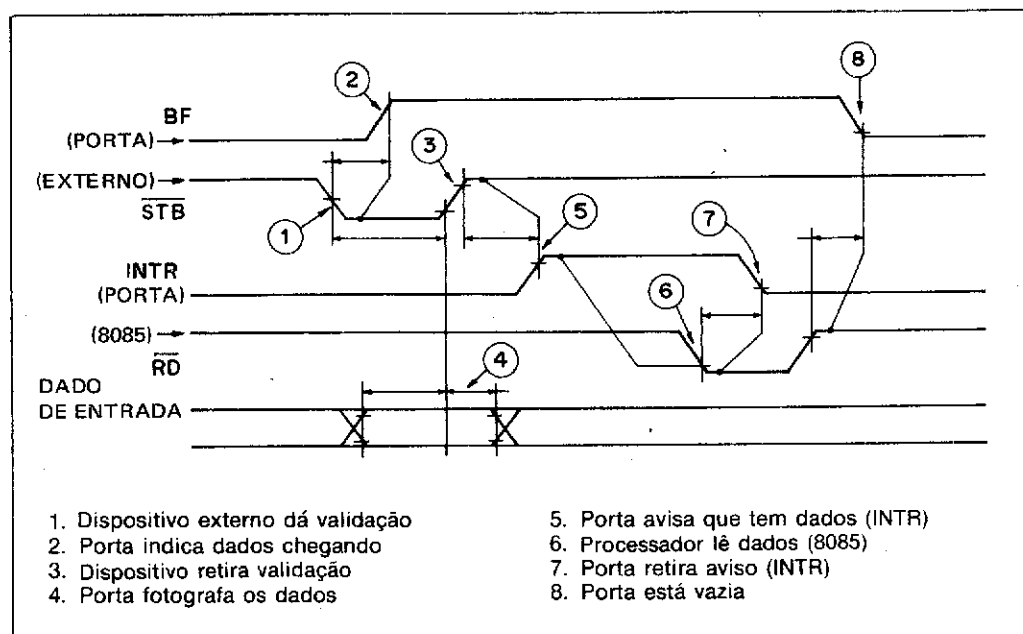



Fig. 5.9 Protocolo de Entrada Controlada

para esta porta. Este sinal (INTR) poderá ser usado para causar uma interrupção no processador. Os sinais BF e INTR serão desligados automaticamente quando o processador fizer uma leitura desta porta.

Note-se que o único sinal de controle gerado externamente é o de validação e que indica que é o momento apropriado para se observar os dados. Os sinais BF e INTR são gerados por circuitos dentro da pastilha.

A Fig. 5.8 ilustra a interação entre a porta de entrada e um dispositivo externo; a Fig. 5.9 ilustra o "protocolo" de sinais.

Numa porta configurada como saída, INTR indicará que a porta está livre e pronta para ser utilizada (isto é, seu conteúdo já foi copiado); e assim pode-se interromper o processador para requisitar um novo byte de dados para saída. Uma operação de escrita na porta desligará o sinal INTR e ligará BF para que o dispositivo externo saiba que existem dados disponíveis.

O dispositivo externo poderá então aplicar a validação $\overline{STB}$ que dirá ao processador que o dado está sendo lido. O início da validação fará com que BF seja desligado e o fim da validação forçará INTR a ser ligado para solicitar um novo byte de dados.

A Fig. 5.10 ilustra a interação entre uma porta de saída e um dispositivo externo; a Fig. 5.11 ilustra o "protocolo" de sinais.

Programação da 8155/8156

A 8155/8156 dispõe de 7 registros internos. Para nos referimos a um desses registros, é necessário que a linha IO/$\overline{M}$ seja 1 (indicando referência a uma porta

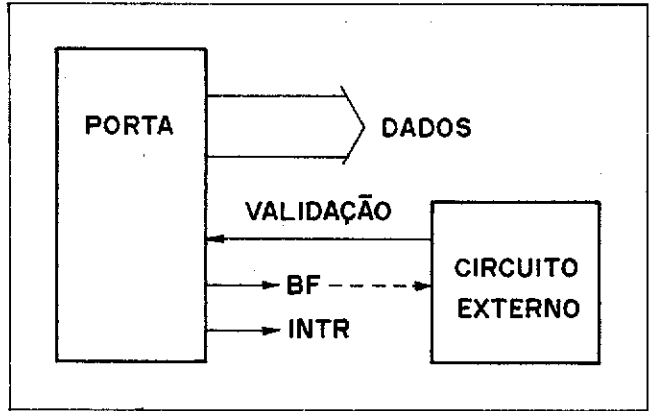



Fig. 5.10 Saída Controlada

1. Processador escreve na porta
2. Porta retira aviso INTR
3. Porta avisa que está cheia
4. Dispositivo externo valida e copia dados
5. Porta está vazia
6. Porta avisa que pode receber novos dados (INTR).

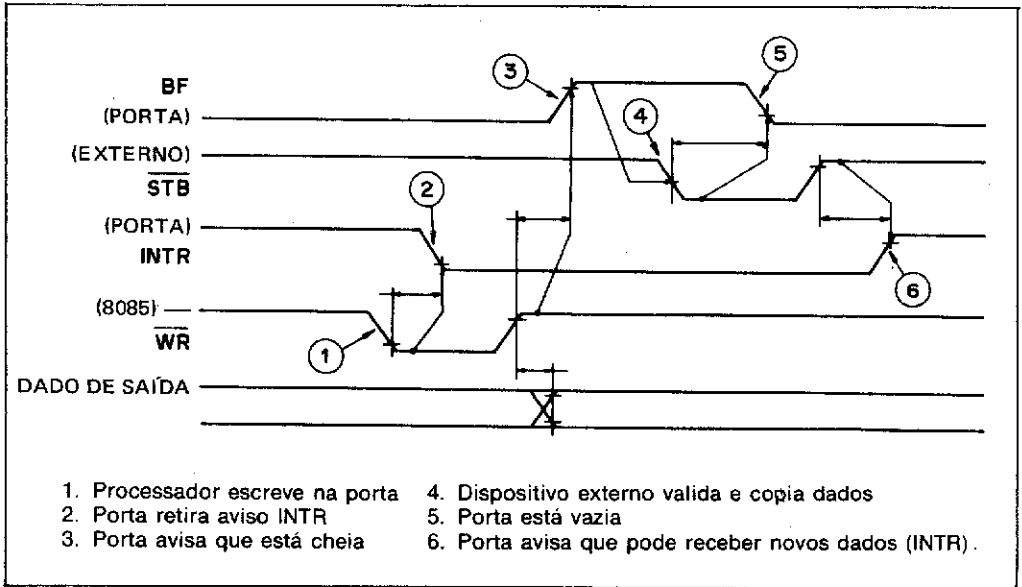



Fig. 5.11 Protocolo de Saída Controlada

e não a uma posição de memória). Os 3 bits de mais baixa ordem selecionarão o registro desejado, como se segue:

000 — registro de comando e registro de estado
001 — porta A
010 — porta B
011 — porta C
100 —}
101 —} registros do relógio.

Note-se que tanto o registro de comando quanto o de estado têm o mesmo endereço (000). Neste caso não há confusão porque o comando é somente *escrito* pelo computador e o estado é somente lido.

Significado e a Utilização dos Registros

Registro de comando. O registro de comando é usado quando o computador precisa especificar alguma ordem (por exemplo, disparar o relógio) ou mudança no tipo de operação de uma porta de entrada/saída.

O formato do registro de comando é ilustrado na Fig. 5.12.

Quando o computador escreve no registro de comando, seus bits são analisados conforme sumarizado na Fig. 5.12.

Bits 7-6 — Referem-se à operação do relógio podendo especificar que o funcionamento permaneça sem alteração, que o relógio pare imediatamente, ou logo que terminar a contagem.

Bits 5-4 — Especificam se os sinais de pedido de interrupção relativos às portas A e B devem ser desabilitados (0) ou habilitados (1).

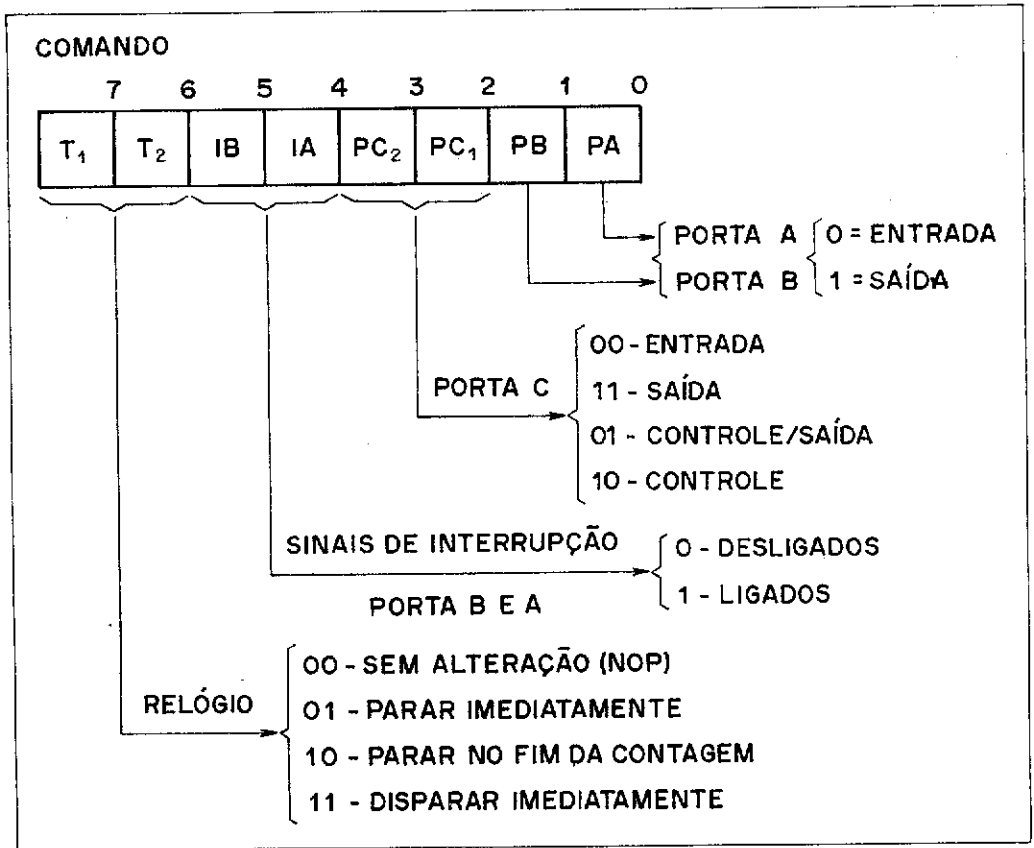



Fig. 5.12 — Registro de comando.

Bits 3-2 — Especificam o modo de funcionamento da porta C. A porta C (que tem 6 bits) pode funcionar como entrada (00), saída (11), pode ser usada para os sinais de protocolo das portas A e B (10) ou pode manter os sinais de protocolo da porta A e os demais 3 bits serem usados como saída (01).
A especificação da porta C através desses dois bits automaticamente implicam na especificação do funcionamento das portas A e B. Isto é, se for dito que C fará o protocolo de A, então subentende-se que a porta A funcionará usando protocolo.

Bits 1-0 — Especificam respectivamente o funcionamento das portas A e B como saída (1) ou como entrada (0).

ESTADO:

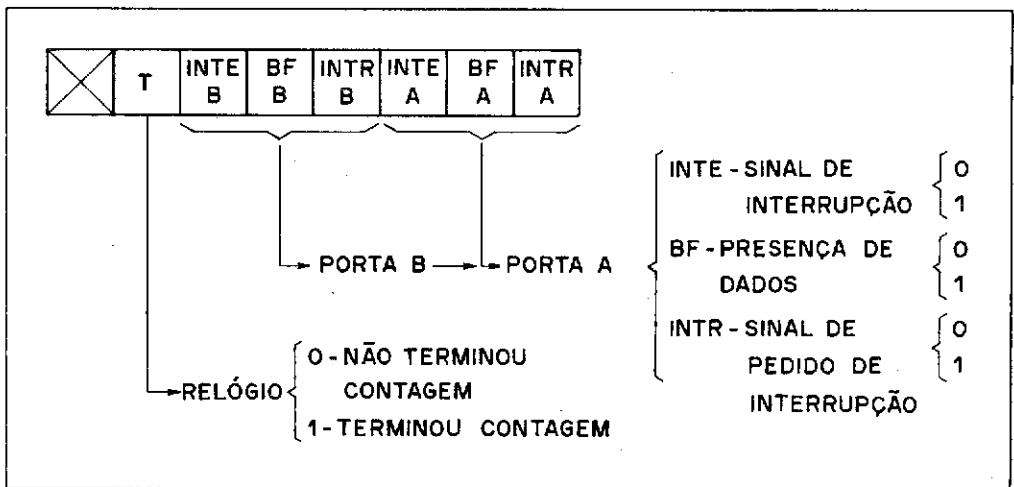



Fig. 5.13

Registro de estado. O formato do registro de estado é como mostra a Fig. 5.13. Quando o computador executa uma instrução de leitura do registro de estado (últimos bits de endereço = 000) as informações de estado são transferidas para o acumulador onde podem ser examinadas.

bit 7 — Não é utilizado.

bit 6 — Ao término da operação do relógio, este bit é ligado, assim permanecendo até que seja efetuada uma leitura do registro de estado ou até que o processador seja reinicializado.

bit 5 — Indica se o sinal de interrupção da porta B está habilitado.

bit 4 — BF (Buffer Full — B) — indica (no caso de operação com protocolo) se foi iniciada uma transferência de dados da porta B.

bit 3 — INTR (Interrupt Request — B) — indica (no caso de operação com protocolo) se foi completada uma transferência da porta B.

bits 2, 1, 0 — Equivalem, respectivamente, aos bits 5, 4, 3 para operações com a porta A.

Portas A, B e C. Os registros cujos endereços terminam em 001, 010 e 011 são, respectivamente, as portas A, B e C. Uma operação de escrita em algum desses registros fará com que o conteúdo das respectivas linhas de saída permaneçam com o valor estabelecido até nova operação de escrita; esta operação é válida quando a porta é configurada para saída. Uma operação de leitura, caso a porta seja configurada como entrada dará, ao computador, o dado presente ou armazenado na respectiva porta.

Relógio. Os dois registros cujos endereços terminam em 100 e 101 são destinados à operação do relógio-despertador. Seus formatos são ilustrados na Fig. 5.14.

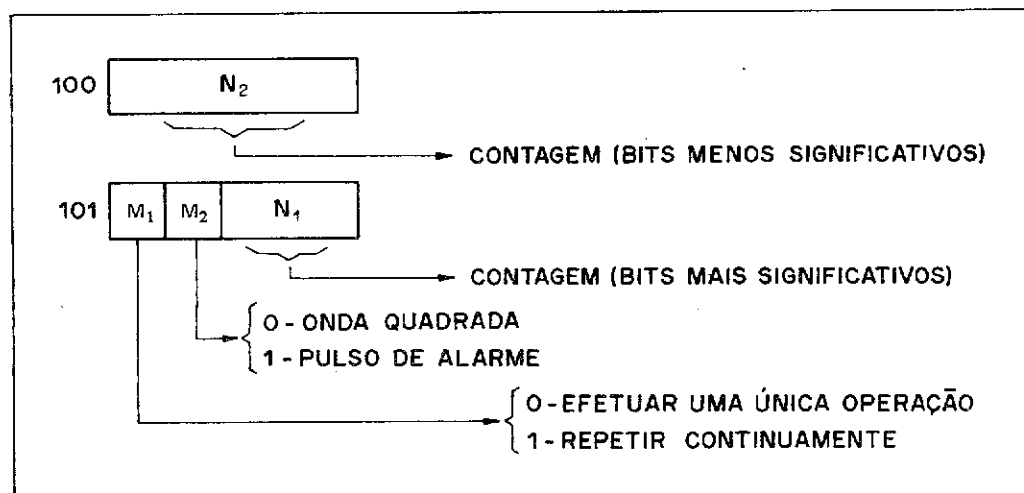



Fig. 5.14 — Registros do Relógio.

O circuito de relógio usa uma linha de entrada (Timer clock) e uma de saída (Timer out).

A operação do relógio resume-se em contar o número de vezes que o sinal de entrada muda para 1. Quando a contagem iguala um número predeterminado o sinal de saída é ativado.

Existem dois parâmetros que especificam o modo de funcionamento do relógio dado pelos bits 7 e 6 do registro cujo endereço termina em 101:

O parâmetro M_2 (bit 7) especifica se, ao final da contagem, o relógio deve parar ($M_2 = 0$) ou repetir (indefinidamente) a mesma operação ($M_2 = 1$).

O parâmetro M_1 (bit 6) especifica a forma do sinal de saída: $M_1 = 0$ faz com que o sinal de saída permaneça em 1 durante a primeira metade da contagem e mude para 0 durante a segunda metade (onda quadrada). $M_1 = 1$ faz com que o sinal de saída permaneça em 1 até o final da contagem onde mudará durante um breve tempo para 0 (Pulso). A Fig. 5.15 ilustra as duas alternativas.

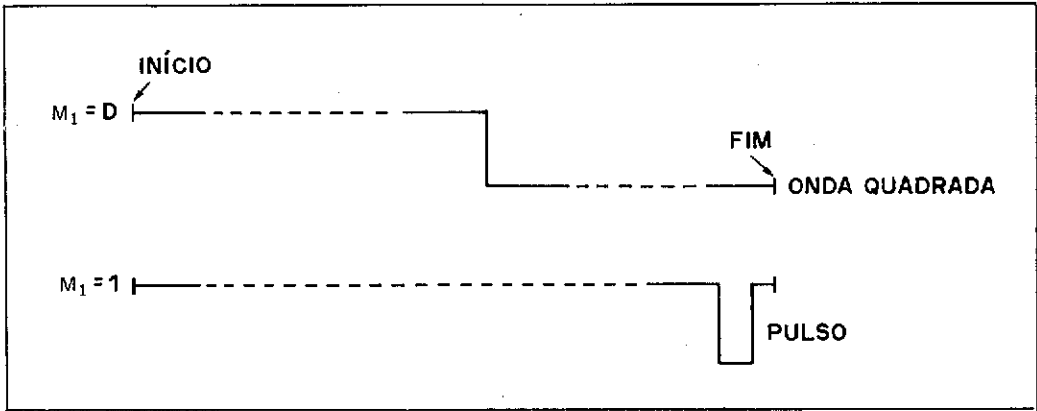



Fig. 5.15 — Onda quadrada e pulso de alarme.

Os demais bits dos registros (N_I e N_2) formarão um número de 14 bits que especificará a contagem. Note-se que, para que o relógio funcione, deve-se carregar os registros e comandar, através do registro de comando, que o relógio seja disparado.

5.3. TÉCNICAS DE ENTRADA/SAÍDA

Para que o computador cumpra a sua finalidade é necessário que ele se comunique com o meio exterior seja obtendo dados de *entrada* que serão usados para cálculos, seja gerando informações de *saída* que irão controlar algum dispositivo ou mesmo prover informações para consulta.

Nesta seção estudaremos as técnicas usuais para entrada e saída de dados que são:

— entrada/saída programada
— entrada/saída por interrupção
— entrada/saída por acesso direto à memória.

Entrada/Saída Programada

Esta é a técnica mais simples de entrada/saída e tem este nome porque é totalmente guiada pelo programador.

Voltando ao exemplo do sinal de pedestre, tínhamos o seguinte fluxograma de entrada:

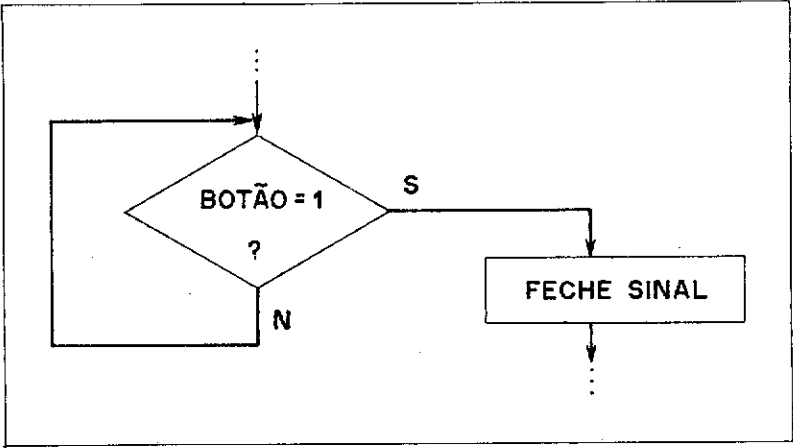



Fig. 5.16 – Teste contínuo.

Do qual podemos tirar as seguintes conclusões:

1. O computador permanece grande parte do tempo testando se o botão está ligado. Em alguns casos isso poderia representar um grande desperdício de tempo do computador. Se este tempo pudesse ser aproveitado de outra forma poderíamos adaptar o fluxograma assim:

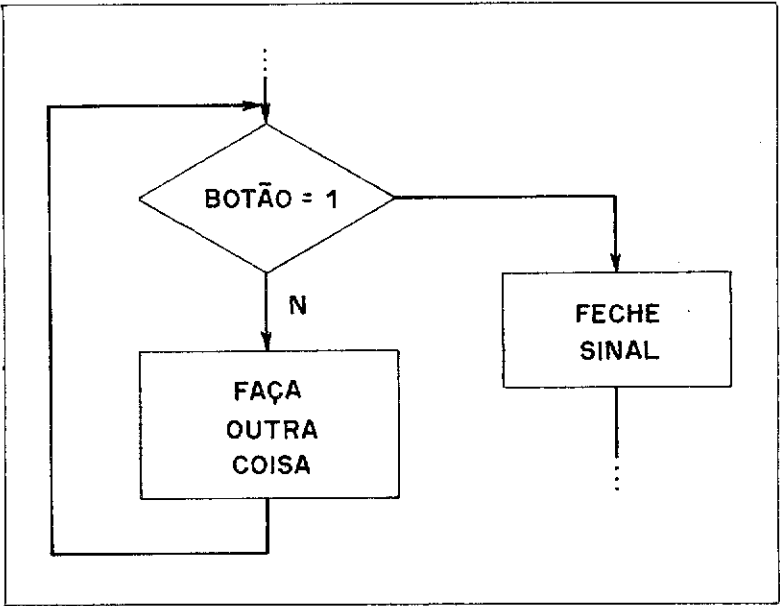



Fig. 5.17 – Teste/processamento.

Correríamos o risco, neste caso, de que, se a "outra coisa" fosse muito demorada, quando voltássemos a testar o botão o sinal não mais estivesse presente (o pedestre aperta e larga o botão). Também ocorreria (embora irrelevante nesse caso) que o pedestre demoraria mais a ser atendido.

2. Se tivessemos muitos sinais a serem controlados teríamos de testar seqüencialmente o pedido de cada um, assim:

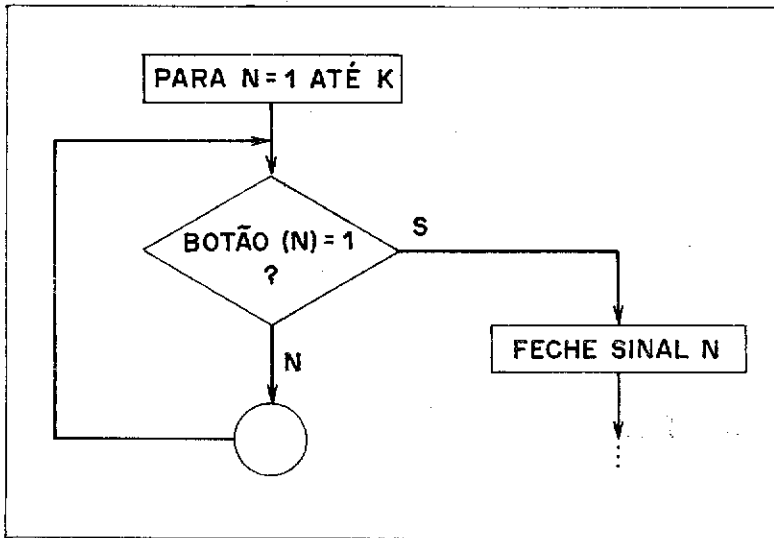



Fig. 5.18 – Teste múltiplo.

Entrada/Saída por Interrupção

Introdução

Nesta modalidade o computador é *avisado* quando é necessário atender a uma entrada/saída. Quando recebe tal aviso, o computador suspende o programa que estava sendo executado (anotando o endereço da próxima instrução para poder continuar o trabalho) e começa a executar um outro programa para cuidar da transferência dos dados. Finda esta, o computador volta a executar o programa anterior.

O procedimento é ilustrado na Fig. 5.19.

Neste caso o tempo para se dar atenção a um dos sinais depende de quantos sinais estejam instalados pois é necessário testá-los um a um.

Para melhor ilustrarmos o que ocorre durante uma interrupção usaremos o seguinte exemplo:

Suponhamos que estamos lendo o jornal e o telefone começa a tocar. Em geral interromperíamos nosso procedimento normal (leitura do jornal) e passa-

S

PEDIDO DE INTERRUPÇÃO ?

N

CHAMADA DE SUB-ROTINA

BUSCA DA PRÓXIMA INSTRUÇÃO

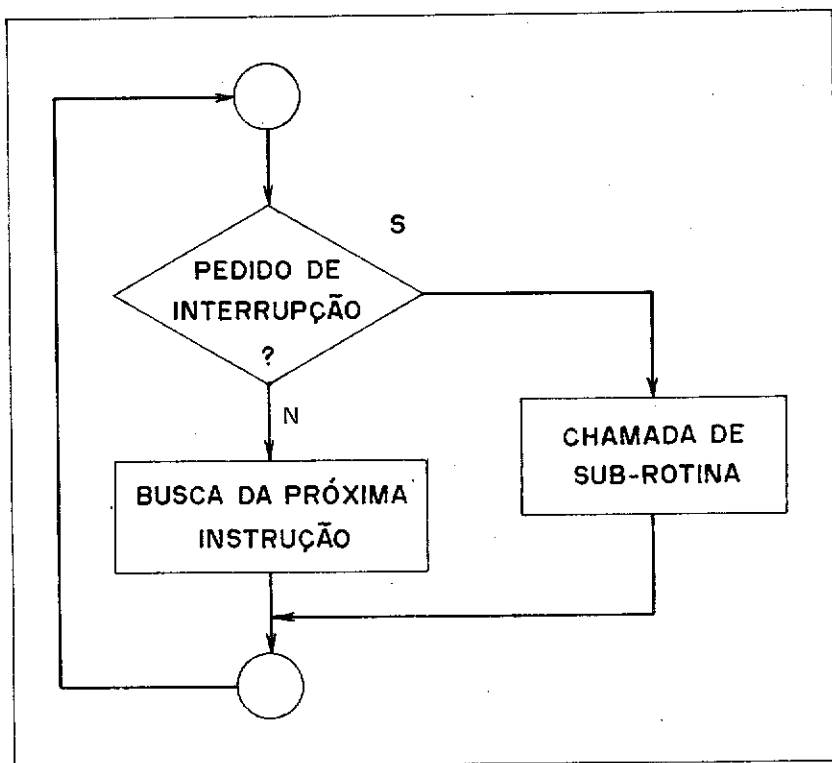



Fig. 5.19 — Interrupção.

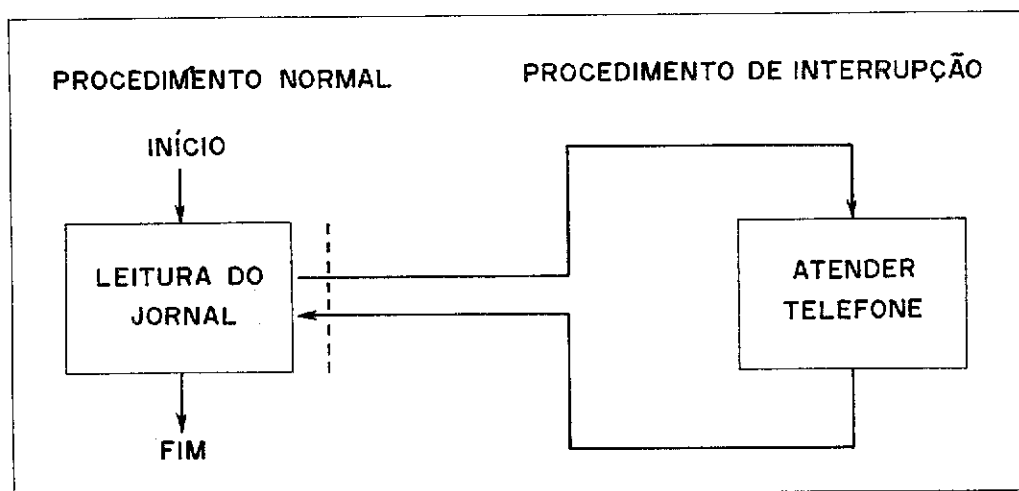



Fig. 5.20 — Interrupção no dia-a-dia.

ríamos a executar o procedimento de interrupção (atender o telefone); após o atendimento deveríamos desligar o telefone e voltar à leitura do jornal. Uma ilustração geral do procedimento pode ser vista na Fig. 5.20.

Assincronismo. O pedido de interrupção (toque do telefone) pode ocorrer a qualquer momento não havendo nenhuma possibilidade de prever em que ponto estaremos da leitura. Daí dizermos que a interrupção é *assíncrona* com o procedimento normal.

O computador também poderá ser interrompido a qualquer ponto da execução do programa (procedimento normal).

Retorno. Após atendermos à interrupção (telefone) devemos retornar ao processamento normal (leitura) exatamente onde nos encontrávamos. Na prática temos uma certa dificuldade em continuar exatamente onde havíamos parado. Para uma maior eficiência poderíamos observar duas regras:

— interromper somente no fim de um parágrafo;
— assinalar, no jornal, o próximo parágrafo.

Terminar a leitura do parágrafo atual, ainda que o telefone tenha começado a tocar, evitaria que desperdiçássemos a parte de leitura já feita (poderia ser no fim de cada artigo, parágrafo, frase, palavra ou letra, isso depende do leitor...).

Assinalar o próximo parágrafo diminuiria o desperdício de tempo ao voltarmos ao texto.

Essas duas regras poderiam aumentar a nossa eficiência sem prejudicar gravemente o atendimento (simplesmente o telefone seria atendido dois ou três toques mais tarde).

No computador duas regras semelhantes são observadas:

— interromper somente no fim de uma instrução;
— o apontador de programa (PC) é armazenado na pilha.

Uma instrução não deve ser interrompida pelo meio pois isso dificultaria sua continuação. (Na verdade, o leitor que acompanhou atentamente o exercício de microprograma não terá dificuldade de perceber que o microprograma testa se há pedidos de interrupção imediatamente antes de iniciar a busca de uma nova instrução.)

O apontador de programa (PC) é tudo que o computador necessita para continuar o procedimento normal.

Procedimento de interrupção. Bem, mas como o computador sabe o que fazer numa interrupção? Quando o telefone toca, quase que intuitivamente temos o ímpeto de atendê-lo. Na verdade esse procedimento já foi observado tantas vezes que nossa mente, ao ouvir o telefone, executa um desvio para esse procedimento.

Para o computador, normalmente, se atribui um endereço determinado para executar em caso de interrupção. Por exemplo, digamos que o endereço de interrupção fosse 0066. Assim, pegaríamos o programa a ser executado quando houvesse interrupção e carrega-lo-íamos a partir deste endereço.

Mascaramento. Outro aspecto interessante é que podemos mascarar (ou seja, ignorar) um pedido de interrupção. Com relação à interrupção do telefone poderíamos, dentre outras coisas, retirá-lo do gancho, tapar os ouvidos etc.

Os computadores normalmente têm um bit armazenado na CPU que indica se as interrupções devem ser atendidas ou mascaradas.

No 8085 esse bit é chamado EI (Enable Interrupt). Se EI = 1 as interrupções são atendidas. O programador pode fazer EI = 1 usando a instrução de mesmo nome, (EI) e pode fazer EI = 0 (mascarando as interrupções) com a instrução DI. EI é sempre desligado automaticamente logo que uma interrupção é reconhecida.

Múltiplas interrupções. É possível ter várias interrupções de vários tipos ocorrendo ao mesmo tempo. O computador normalmente associa uma prioridade de atendimento quando mais de um tipo ocorre simultaneamente (telefone e alarma de incêndio...). Também pode ser necessário, em alguns casos, suspender o procedimento de interrupção para atender outra interrupção mais prioritária. Praticamente todos os computadores permitem esse procedimento.

Interrupção no 8085

No 8085 existem dois modos de interrupção:

— Direto (Restart)
— Pedido (INTR)

Apesar de que os dois modos são muito semelhantes discutiremos detalhadamente o modo direto que é mais simples.

Dentro do processador existem dois registros usados exclusivamente para interrupções mostrados na Fig. 5.21. Registro de máscaras (RM) e registro de pedidos (RP). O registro de pedidos (RP) guarda informações referentes aos

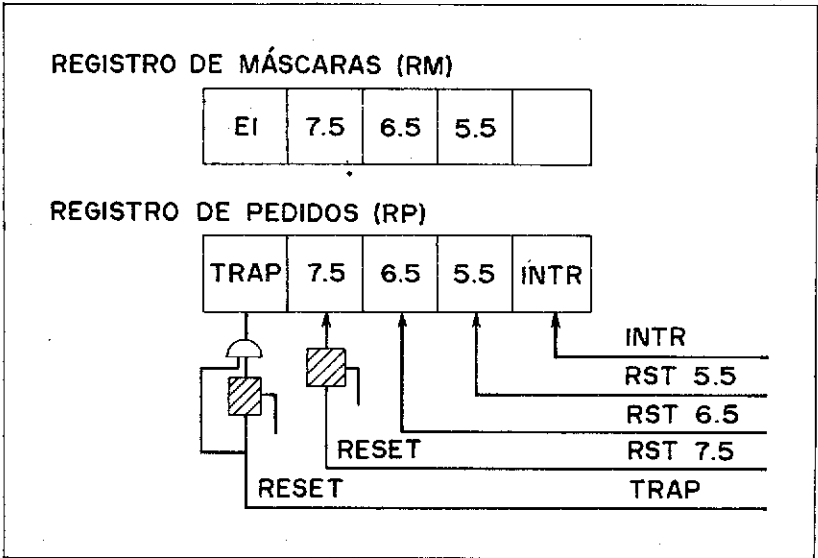



Fig. 5.21 — Registros de Interrupção.

pedidos de interrupções. Cada bit deste registro está ligado a uma das linhas de interrupção que provém do exterior da pastilha. Os bits 6.5, 5.5 e INTR estão diretamente ligados às linhas externas e, portanto, serão iguais a zero ou um dependendo do estado dessas linhas. Os bits TRAP e 7.5 estão também ligados às respectivas linhas porém através de circuitos adicionais. O circuito representado por um quadrado é um "flip-flop" armado quando a linha de entrada muda do estado zero para o estado um; isto é, considerando sua saída inicialmente zero, quando a linha externa mudar de zero para um, a saída do "flip-flop" passará a ser um. Caso a linha de entrada volte a zero, ainda assim o "flip-flop" continuará com a saída em um até que o processador comande sua reinicialização (Reset). Note-se que, se a linha externa continuar em "1" após a reinicialização, a saída do "flip-flop" não passará a ser "1" pois somente a *mudança* de "0" para "1" é que liga esse circuito.

Existem diversas situações onde é conveniente usar esse tipo de pedido de interrupção (7.5) basicamente por duas razões:

— Evitar perda de pedidos
(O pedido de interrupção ocorre durante um tempo curto.)

No sinal de tráfego, por exemplo, o pedestre aperta e larga o botão.

Se o dispositivo acionador (no caso o pedestre) fosse muito rápido seria possível que o computador perdesse o pedido caso o mesmo fosse feito durante um tempo em que o computador não atendesse interrupções (durante uma instrução, durante um mascaramento etc.). Usando este "flip-flop" na entrada, o pedido seria guardado até ser atendido.

— Evitar duplicação de pedidos
(O pedido de interrupção não é desligado imediatamente após o atendimento.)

Ainda pensando no exemplo do sinal de tráfego, o pedestre poderia continuar pressionando o botão mesmo depois de ser atendido causando uma série de interrupções com um mesmo pedido. Isto também seria ilustrado pensando numa leitora de fita de papel que usássemos o sinal de furo de tração diretamente para pedir a interrupção. Se o computador lesse o dado perfurado rapidamente, o pedido de interrupção poderia continuar (furo de tração ainda não acabou de passar) causando nova interrupção e leitura do mesmo dado lido anteriormente.

No caso da linha "TRAP" a saída do "flip-flop" passa ainda por um "E" lógico, sendo portanto necessário que a saída do "flip-flop" esteja ligada "E" que o pedido esteja ligado para que o mesmo seja registrado.

Assim, o projetista usará o tipo de interrupção que for mais conveniente para o seu caso.

O registro de pedidos guarda os pedidos de interrupção, porém o processador somente atenderá tais pedidos observando o estado das máscaras como se segue:

— O bit EI (habilitação de interrupções) comanda todas as interrupções exceto TRAP. Se EI = 0 nenhuma destas é atendida.

— O pedido de TRAP é de caráter catastrófico e será atendido sempre, independente do estado de qualquer máscara.

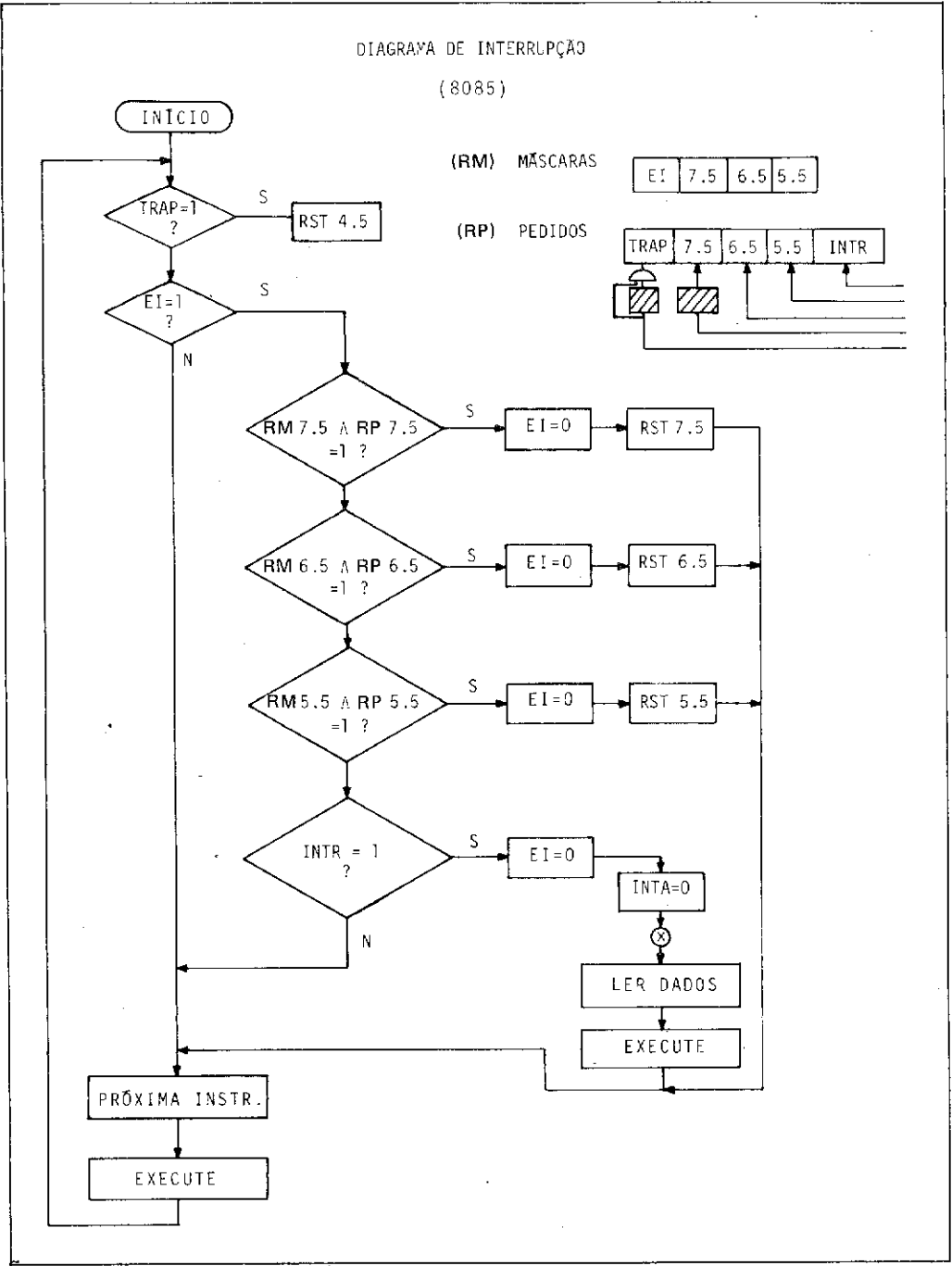



Fig. 5.22 — Procedimento de Interrupção no 8085.

— Os bits 7.5, 6.5 e 5.5 do registro de máscaras comandam, individualmente os pedidos 7.5, 6.5 e 5.5, respectivamente. Ou seja: para que o pedido 7.5 seja atendido é preciso que a máscara 7.5 esteja desligada (*igual a zero*).

A Fig. 5.22 dispõe, na forma de um fluxograma, o mecanismo de atendimento de interrupções.

As interrupções causadas por TRAP, RST 7.5, RST 6.5 e RST 5.5 fazem com que o *hardware* execute uma instrução RST (restart) ocasionando uma chamada de sub-rotina para endereços determinados de memória (ver tabela da Fig. 5.23).

	ENDEREÇO DE DESVIO		
PEDIDO	DECIMAL	HEXA	PRIORIDADE
TRAP (RST 4.5)	36	24	1
RST 7.5	60	3C	2
RST 6.5	52	34	3
RST 5.5	44	2C	4

Fig. 5.23 — Endereços de Interrupções.

No caso do reconhecimento simultâneo de mais de um pedido, a ordem de prioridade é: TRAP (maior prioridade), 7.5, 6.5, 5.5 e INTR (menor prioridade).

A interrupção causada por INTR não causa automaticamente um desvio para sub-rotina. Nesta interrupção o computador fica, literalmente, "sem saber o que fazer". O procedimento normal é interrompido e a busca da próxima instrução ocorre de modo peculiar: ao invés de mandar o endereço do PC para a memória e aguardar a instrução na via de dados, o processador envia apenas um sinal $\overline{INTA}$ (reconheço interrupção) e aguarda a instrução (colocada por um dispositivo externo) na via de dados.

Entrada/Saída por Acesso Direto (DMA)

Apesar de que a transferência por interrupção diminua bastante o tempo desperdiçado pelo processador, ainda exige que o mesmo trabalhe bastante para cada transferência. Isto porque o processador é responsável por todo o processo de transferência.

Uma modalidade bastante comum de transferência requer que os dados presentes em um dispositivo de entrada sejam transferidos para endereços contínuos de memória à medida que forem aparecendo. Geralmente, o processador não necessita de examinar os dados até que toda a seqüência esteja pronta. Tal é o caso com leitura de cartões, fita, discos e, de modo análogo, para saídas em fita, disco e impressora.

Para esse tipo de transferência foi idealizado o processador DMA — acesso direto à memória, conforme ilustra a Fig. 5.24.

O processador DMA deve ser ligado às vias de comunicação e ao periférico a ser controlado. O DMA contém dois registros importantes: o registro de endereços (que pode ser incrementado) e o registro de contagem (que pode ser

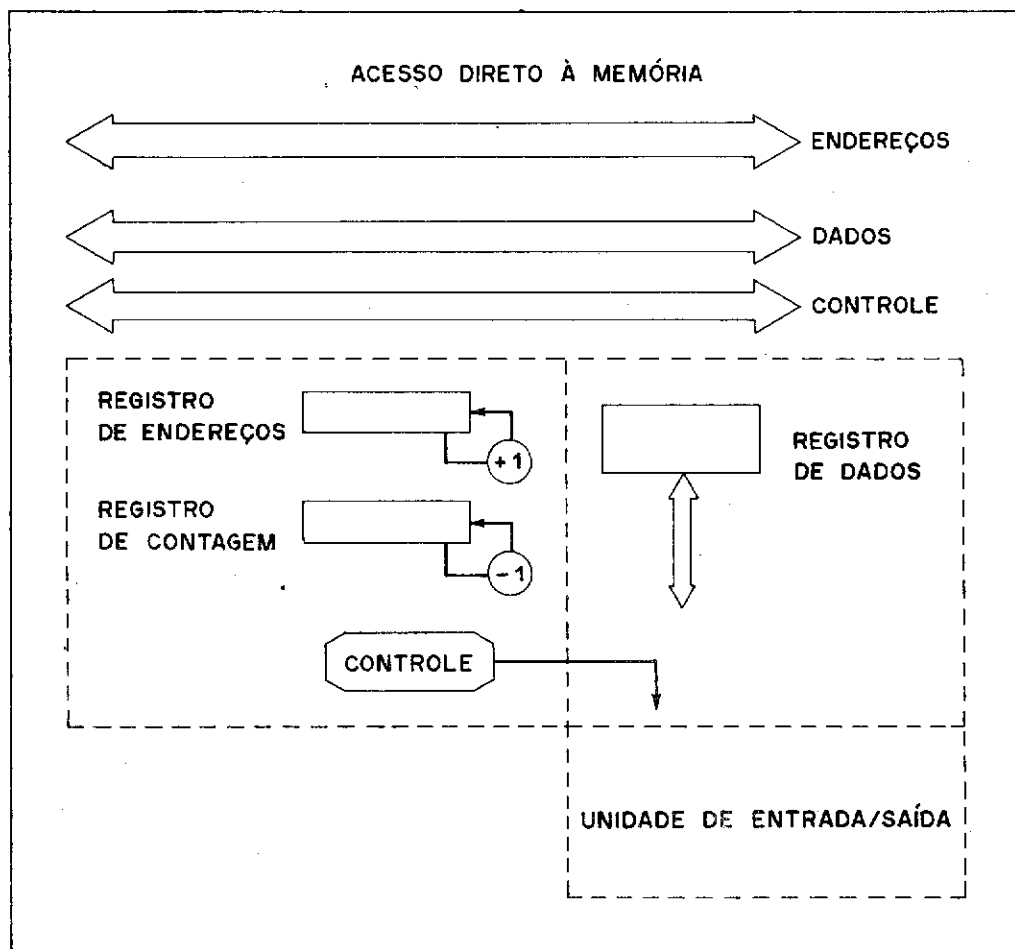



Fig. 5.24 – Processador DMA.

decrementado). Esses registros também podem ser carregados através de instruções do processador.

Podemos considerar três fases distintas durante a operação do DMA: Inicialização, transferência e finalização.

Inicialização. Durante a inicialização os registros do DMA são carregados pelo programa em execução no microprocessador. Dessa forma o DMA é instruído a transferir um determinado número de dados (constante do registro de contagem) de/para uma área de memória a partir de determinado endereço (constante do registro de endereço). Uma vez que o DMA está inicializado, o microprocessador continua operando normalmente enquanto o DMA aguarda a chegada de dados.

Transferência. Quando algum dado se torna disponível, o equipamento envia um sinal de controle ao DMA. (Note-se que, na transferência por interrupção, o periférico envia este sinal de controle ao processador que é interrompido para gerenciar a transferência.)

O DMA então será responsável por transferir o dado sem a interferência do processador. Ora, o DMA sabe que o dado se encontra no periférico e sabe qual o endereço de memória onde o mesmo deverá ser armazenado; assim, ele mesmo (DMA) coloca o endereço na via de endereços, comanda o periférico para colocar o dado na via de dados e gera os sinais de controle apropriados.

Obviamente as vias não podem ser usadas ao mesmo tempo pelo DMA e pelo processador, isto será visto adiante.

A memória, que está permanentemente monitorando as vias, simplesmente observa que o dado deve ser gravado em determinado endereço e obedece (a memória não distingue se o comando veio do processador ou do DMA).

Após esta operação o DMA incrementa o registro de endereços (para que o próximo dado seja gravado seqüencialmente) e decrementa o registro de contagem; se este continuar positivo o DMA executará nova transferência quando o próximo dado estiver disponível. Caso a contagem seja zero, o DMA passará ao estado de finalização.

Finalização. Após transferir o último dado de uma série (registro de contagem fica igual a zero), o DMA suspende qualquer outra transferência e envia um sinal ao processador indicando que a transferência está completa. Em geral este sinal deve causar uma interrupção no processador.

Uso das vias. Conforme foi mencionado, DMA e processador não podem usar as vias ao mesmo tempo. Para evitar conflitos, existem dois sinais de controle trocados entre o DMA e o processador; normalmente estes sinais são chamados HOLD e HLDA.

Quando o DMA deseja usar as vias, ele envia o sinal HOLD ao processador e aguarda, como resposta, o sinal HLDA. O processador, se não estiver usando as vias, cede-las-á imediatamente; no entanto, caso ele esteja efetuando alguma transferência, esta prosseguirá até o fim e só então as vias serão cedidas.

Ceder as vias significa que o processador se abstém de usá-las até que o sinal HOLD seja retirado; as vias que saem do processador permanecerão em estado indeterminado (alta impedância) durante este período.

É também oportuno ressaltar que, diferentemente do que ocorre com as interrupções, uma transferência DMA pode ser atendida durante a execução de uma instrução.

Interferência com o processador. Em geral, uma transferência DMA tem prioridade sobre uma transferência do processador. Isto significa que o processador sempre cede as vias para o DMA usar; resultando daí, que o processador sofrerá um certo atraso no seu trabalho devido à interferência do DMA.

Para cada dado transferido, o atraso máximo causado será de 1 ciclo de memória (portanto, da ordem de 450 ns). Dizemos atraso máximo porque é possível que o processador esteja executando operações que não dependem de transferência nas vias podendo, neste caso, trabalhar normalmente.

5.4. MONTAGEM DO SISTEMA MÍNIMO

A Fig. 5.25 ilustra uma possível montagem do sistema mínimo em discussão. Neste tipo de montagem não foi necessário usar um decodificador; ao invés

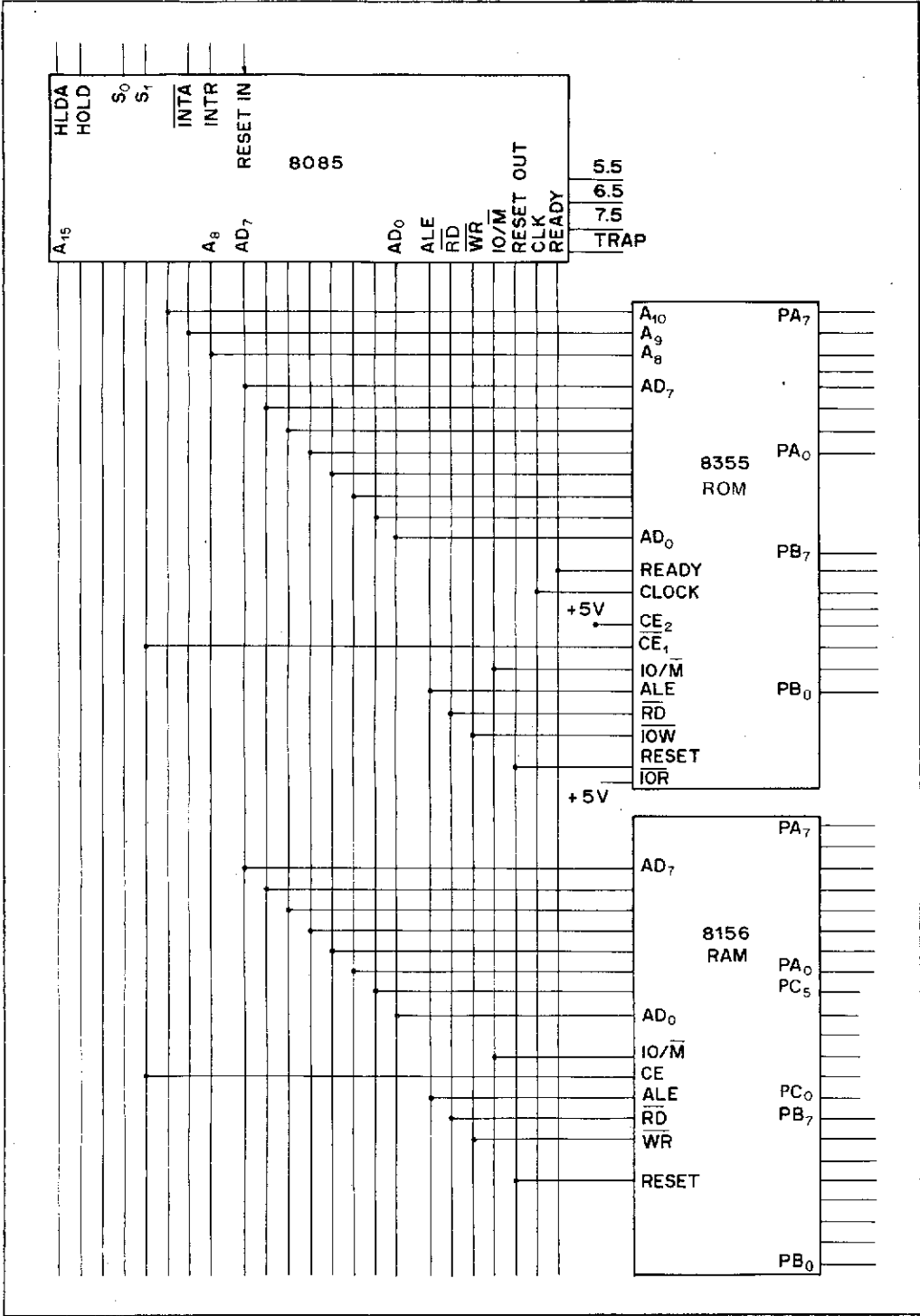



Fig. 5.25 – Sistema Mínimo.

disso usamos uma das linhas de endereço (no caso A_{11}) para selecionar a ROM (quando $A_{11} = 0$) ou a RAM (quando $A_{11} = 1$). Qualquer outra linha livre poderia ser usada com esta finalidade; porém, usando-se A_{11} os endereços da ROM e RAM serão contínuos.

Em princípio, vamos associar com a ROM os endereços de 0000 a 07FF e com a RAM os endereços de 0800 a 08FF; porém, de acordo com o esquema utilizado, qualquer endereço em que $A_{11} = 0$ selecionará a ROM, dessa forma os endereços 0000 0000 0000 0000, 0001 0000 0000 0000, 0011 0000 0000 0000 etc., todos se referem à mesma posição de memória, ou seja, uma mesma posição pode ser referenciada por mais de um endereço. O mesmo ocorre com a memória RAM.

Se ao invés de selecionarmos as memórias com uma linha de endereços (seleção linear) usássemos decodificadores (seleção decodificada) poderíamos ter um único endereço para cada posição de memória; no entanto, isso não compensaria a despesa adicional neste sistema.

RESUMO DO SISTEMA MÍNIMO
ENTRADA/SAÍDA INDEPENDENTE

NÚMERO DA PORTA	LOCALIZAÇÃO	DESCRIÇÃO
0	ROM	PORTA "A" da ROM
1	ROM	PORTA "B" da ROM
2	ROM	REGISTRO DE DIREÇÃO DA PORTA "A"
3	ROM	REGISTRO DE DIREÇÃO DA PORTA "B"
4		
5		
6		
7		
8	RAM	REGISTRO DE COMANDO OU DE ESTADO
9	RAM	PORTA "A" da RAM
10	RAM	PORTA "B" da RAM
11	RAM	PORTA "C" da RAM
12	RAM	RELÓGIO, parte menos significativa
13	RAM	RELÓGIO, parte mais significativa

Fig. 5.26 – Portas de E/S no Sistema Mínimo.

Os registros no interior da ROM e RAM também ficam associados a endereços: as portas A e B e os registros de direção A e B da ROM poderão ser referenciados como portas 0, 1, 2 e 3 e os registros no interior da RAM poderão ser referenciados como portas 8, 9, 10, 11, 12 e 13 (sugere-se que o leitor procure entender porque estes endereços se referem a essas portas) (ver tabela da Fig. 5.26).

Entrada/Saída Mapeada na Memória

Uma técnica bastante usada atualmente é a de se usar endereços de memória para referenciar as portas de entrada/saída. A idéia consiste em "enganar" tanto o processador como as portas, sendo uma possível solução o esquema proposto na Fig. 5.27.

Neste esquema ligamos a linha A_{15} às entradas $IO/\overline{M}$ tanto da ROM quanto da RAM. Quando o processador faz referência a algum endereço em que $A_{15} = 1$, as pastilhas de memória irão pensar que se trata de uma entrada/saída (já que a entrada $IO/\overline{M} = 1$). Por outro lado, o processador irá crer que está dialogando com a memória e não com uma porta. A grande vantagem desta técnica está justamente em usar as instruções de memória que são muito mais variadas e poderosas que as de entrada/saída. Uma desvantagem, que não é relevante neste exemplo, é que menos endereços de memória ficam disponíveis já que alguns endereços estarão associados às portas.

Uma Aplicação: Controlador de Sinal de Pedestre

Uma aplicação simples para o sistema mínimo em discussão seria implementar o controle do sinal de pedestre apresentado no Cap. 1. Para este controle necessitamos apenas de uma linha de entrada (botão de pedestre) e duas de saída (lâmpada vermelha, lâmpada verde).

Podemos escolher à vontade em que linhas e em que portas ligar estes sinais; poderíamos; por exemplo, usar apenas a porta "A" da ROM mas, para maior simplicidade, vamos usar o bit 0 da porta "A" para a entrada (botão) e os bits 1 e 0 da porta "B" para as saídas conforme indica a Fig. 5.28.

Resta-nos apenas fazer o programa para realizar o controle, conforme sugerido no Cap. 1. Um pequeno detalhe que precisamos observar refere-se à programação das portas pois, como já sabemos, é necessário explicar quais linhas serão de entrada e quais de saída.

Porta A — Apenas o bit 0 será usado como entrada; portanto, o Registro de Direção dessa porta deverá conter um valor 0 na posição do bit 0. Os demais bits poderiam ser 0 ou 1, vamos escolher deixá-los em 0. Assim, esse registro deverá conter oito zeros.

Porta B — Os bits 0 e 1 serão usados como saída; portanto, seu Registro de Direção deverá conter o valor 1 nas posições 0 e 1. Os demais bits poderiam ser 0 ou 1, vamos escolher deixá-los em 0.

Como já foi visto, o Registro de Direção das portas A e B receberão os endereços 2 e 3.

Registros de Direção:

0000 0000	da Porta A
0000 0011	da Porta B

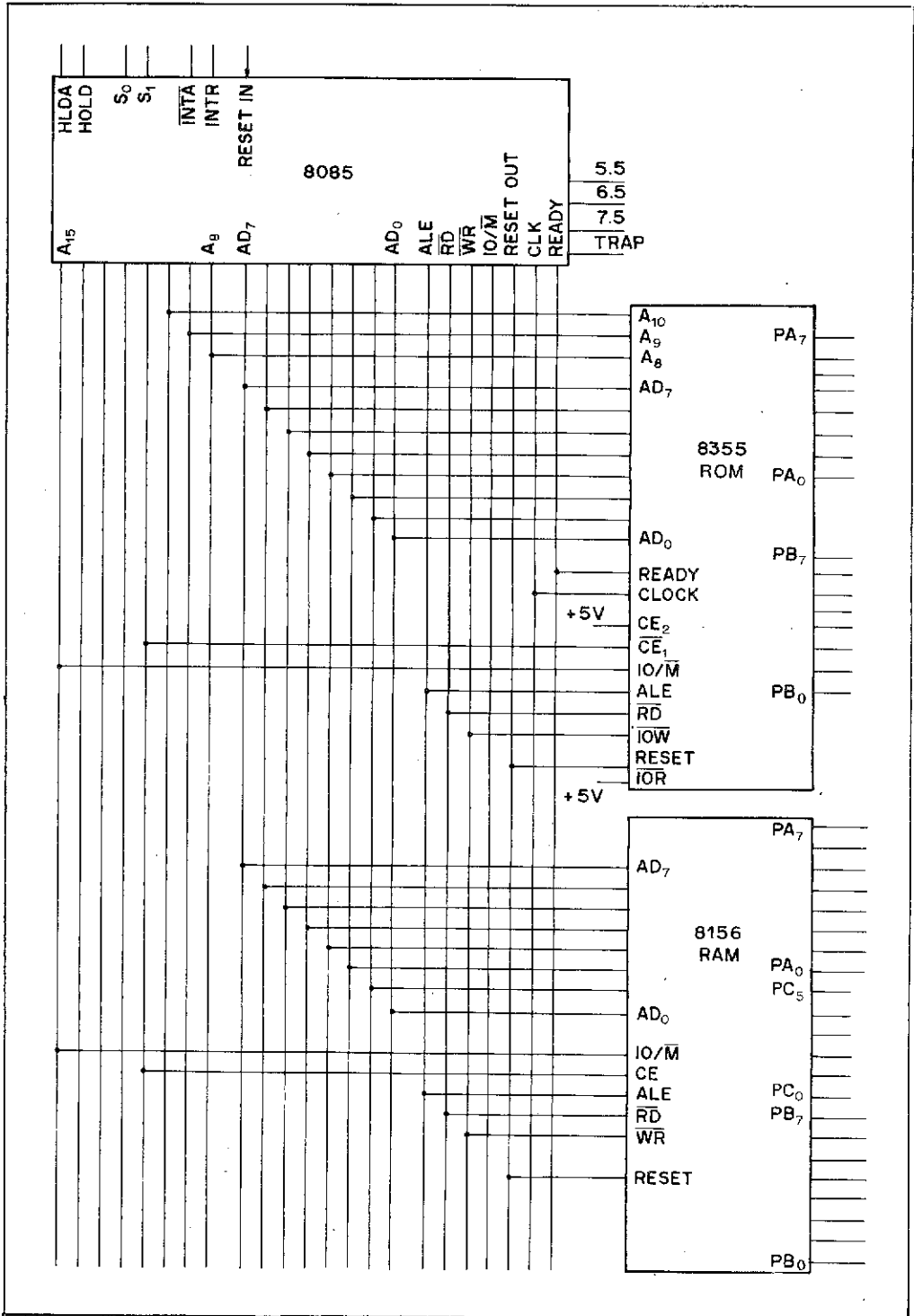



Fig. 5.27 — E/S mapeada na memória.

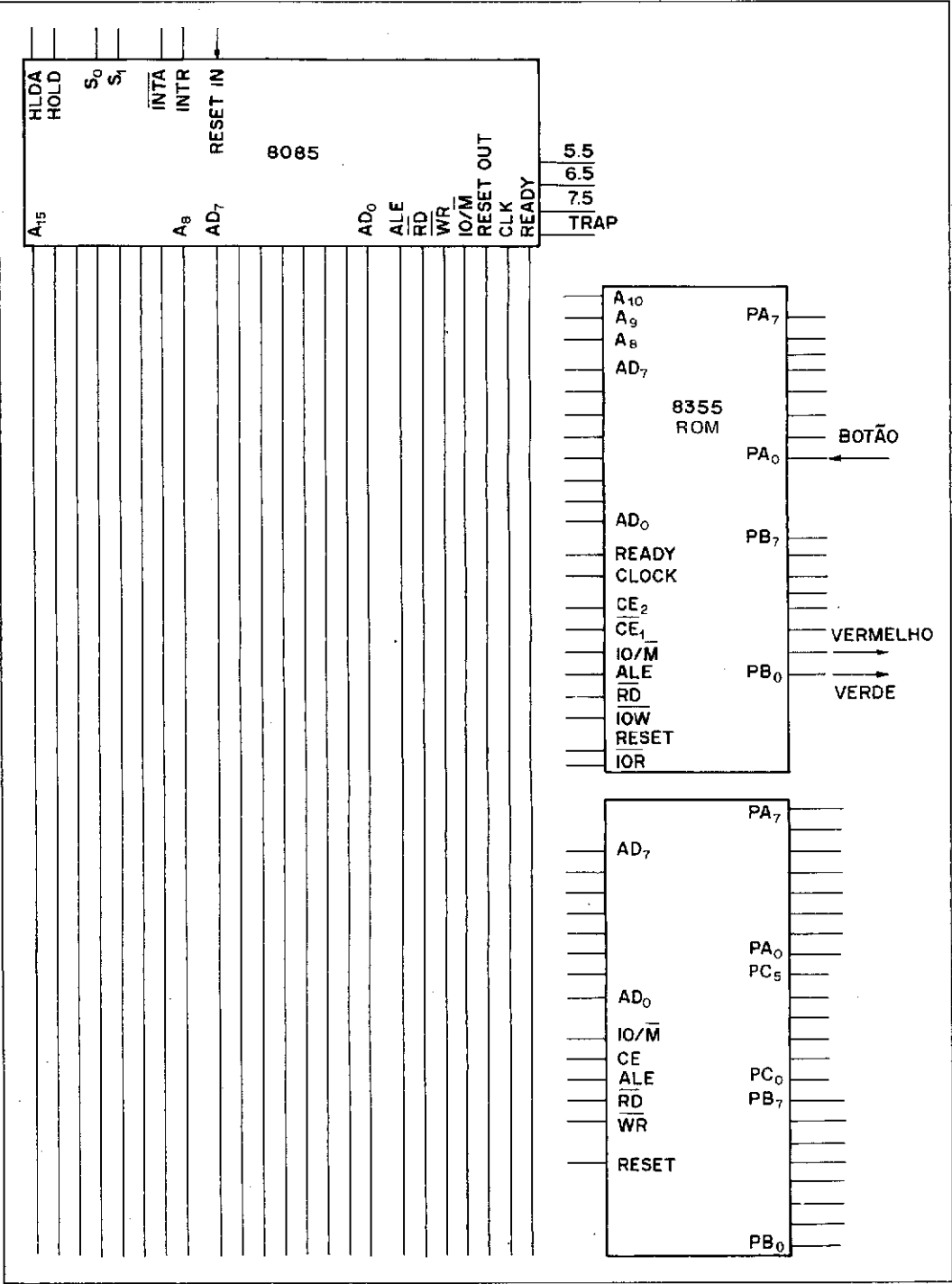



Fig. 5.28 — Ligação dos sinais externos.

Para que esses valores sejam corretamente carregados, é necessário executar um trecho de programa como, por exemplo:

```
MVI A, 00     (carrega zero no acumulador)
OUT 02        (coloca no registro direção de A)
MVI A, 03     (carrega 03 no acumulador)
OUT 03        (coloca no registro direção de B)
```

Após a execução desse programa, poderemos saber qual o estado do botão executando uma leitura da porta A (isto é, IN 00), pois os valores presentes à entrada da porta serão trazidos para o acumulador. Analogamente, podemos controlar o estado das lâmpadas carregando o acumulador com a configuração de bits desejada e executando uma saída para a porta B (isto é, OUT 01).

O programa da Fig. 5.29 segue o fluxograma indicado no Cap. 1.

O programa deve ser facilmente inteligível para o leitor que deverá ter notado algumas chamadas para a sub-rotina de nome ESPR. Esta sub-rotina deverá executar o controle de tempo consumindo 10 segundos cada vez que for chamada. Existem várias maneiras de se fazer o controle de tempo; a mais simples (e menos elegante) consiste em executar um trecho de programa cuja duração seja aproximadamente igual ao tempo desejado.

```
ESPR:   MVI   C, Y1
X1:     MVI   D, Y2
X2:     MVI   E, Y3
X3:     DCR   E
        JNZ   X3
        DCR   D
        JNZ   X2
        DCR   C
        JNZ   X1
        RET
```

Neste exemplo, vemos que as instruções DCR E e JNZ X3 são executadas $Y1 \times Y2 \times Y3$ vezes. Sabendo que o tempo consumido por essas instruções é de 7 us* poderemos calcular o número de vezes que devemos executá-las dividindo o tempo total desejado (10 segundos) pelo tempo de cada execução (7 us), donde resulta 1.400.000 (aproximadamente) poderemos, por exemplo, fazer $Y1 = 7$, $Y2 = 100$ e $Y3 = 200$.

Estamos desprezando outras instruções que serão executadas e usando um valor aproximado. Como o controle de tempo não necessita de muita precisão (não faz mal que 10 segundos sejam na verdade 9:45 ou 10:15) essa aproximação é aceitável; caso uma maior precisão fosse exigida todas as instruções deveriam ser consideradas.

---

* O tempo de cada instrução pode ser calculado multiplicando-se o número de períodos (ver Apêndice A) pela duração de cada período; tipicamente 330 ns.

```
                    ;*******************************
                    ;         CONTROLE DE SINAL DE PEDESTRE
                    ;*******************************
                    ;
                    ;*******************************
                    ;         INICIALIZACAO DAS PORTAS DE E/S
                    ;*******************************
0000 3E00                     MVI   A,0    ;CARREGA ZERO NO ACUMULADOR
0002 D302                     OUT   2      ;COLOCA NO REGISTRO DE DIRECAO A
0004 3E03                     MVI   A,3    ;CARREGA 3 NO ACUMULADOR
0006 D303                     OUT   3      ;COLOCA NO REGISTRO DE DIRECAO B.
                    ;**********************
                    ;         PROGRAMA DE CONTROLE
                    ;**********************
0008 3E01           INIC:     MVI   A,1
000A D301                     OUT   1      ;ACENDE LAMPADA VERDE
000C DB00           LER:      IN    0      ;LE ESTADO DO BOTAO
000E 3D                       DCR   A
000F FA0C00                   JM    LER    ;SE NAO HOUVER PEDESTRE VOLTA
0012 3E03                     MVI   A,3
0014 D301                     OUT   1      ;ACENDE VERDE E VERMELHO
0016 CD2900                   CALL  ESPR    ;ESPERA 10 SEGUNDOS
0019 3E02                     MVI   A,2
001B D301                     OUT   1      ;ACENDE LAMPADA VERMELHA
001D 0603                     MVI   B,3
001F CD2900         LOOP:     CALL  ESPR   ;ESPERA 10 SEGUNDOS
0022 05                       DCR   B
0023 F21F00                   JP    LOOP   ;VOLTA PARA LOOP 3 VEZES
0026 C30800                   JMP   INIC
                    ;
                    ;
                    ;
                    ;
                    ;**********************
                    ;          ROTINA DE ESPERA DE 10 SEGUNDOS
                    ;**********************
0029 0E07           ESPR:      MVI   C,7   ;CARREGA C COM Y1
002B 1664           X1:        MVI   D,100 ;CARREGA D COM Y2
002D 1EC8           X2:        MVI   E,200 ;CARREGA E COM Y3
                    ;
                    ;
002F 1D             X3:        DCR   E     ;*EXECUTA ESSAS 2 INSTRUCOES
0030 C22F00                    JNZ   X3    ;* Y1.Y2.Y3 VEZES
                    ;
                    ;
0033 15                        DCR   D
0034 C22D00                    JNZ   X2
                    ;
0037 0D                        DCR   C
0038 C22B00                    JNZ   X1
                    ;
003B C9                        RET         ;RETORNA AO PROGRAMA
003C                           END   INIC
```

Fig. 5.29 — Controle do Sinal de Pedestre.

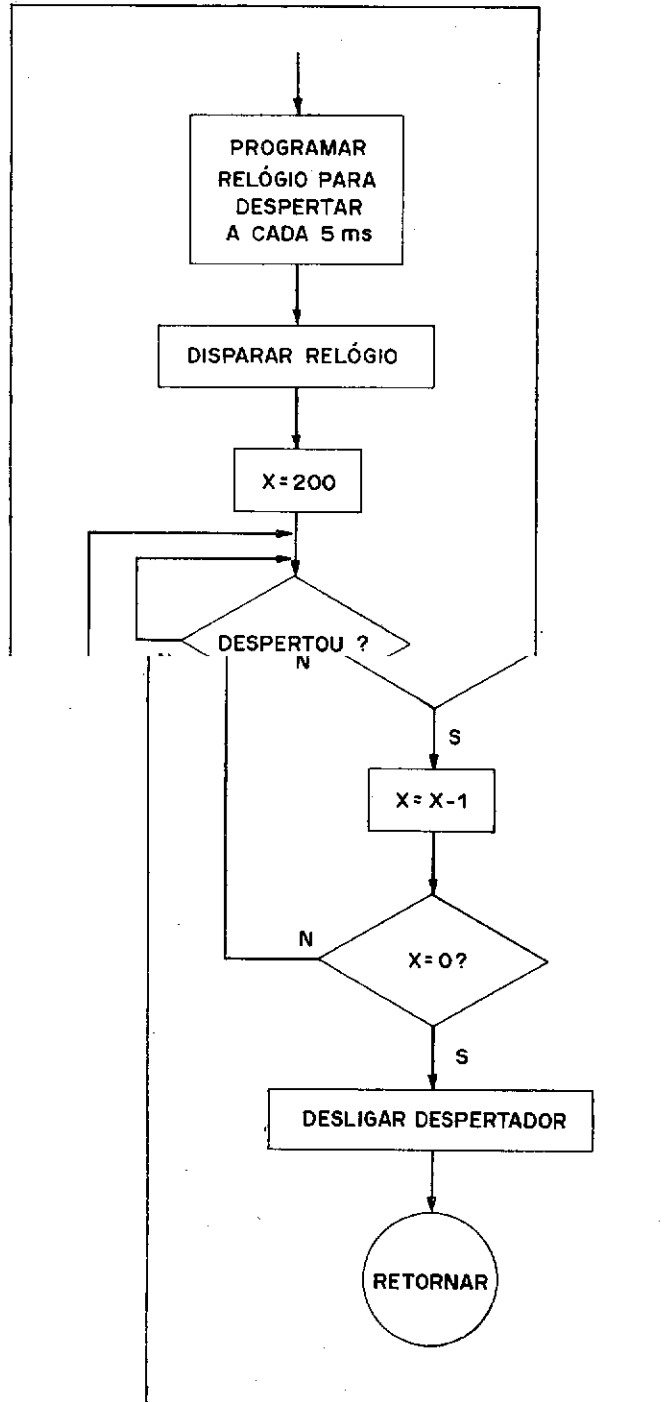



Fig. 5.30 — Fluxograma de uso do despertador.

Comentários desta solução. O programa e a configuração de *hardware* apresentados resolvem o problema do controlador de sinais. O controle de tempo por *software* atende às necessidades, porém, é um tanto deselegante já que envolve a necessidade do programador saber o tempo de execução de instruções. O uso de um relógio/despertador é uma solução mais elegante, mas, normalmente, envolve um acréscimo no custo que seria desnecessário já que o problema pode ser resolvido somente com o *software*. Porém, já que estamos usando o sistema mínimo que inclui um despertador, nada mais razoável do que usá-lo.

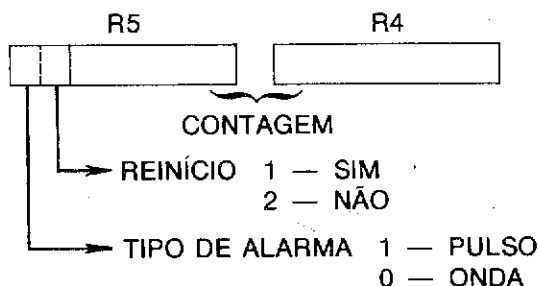
Controle de Tempo com Despertador

Como já foi visto anteriormente, o despertador do 8085 usa um registro que pode ser decrementado cada vez que surge um sinal na entrada. No nosso caso, o único sinal disponível é o próprio relógio do processador que, nos modelos comuns, gera um sinal a cada 330 ns.

Como já sabemos, o registro do despertador usa 14 bits podendo, portanto, armazenar um número de 0 a 16384. Gostaríamos de poder usar um número maior já que serão necessários 10 s/330 ns = 30×10^6 sinais de 330 ns para fazer 10 segundos. Assim, nosso despertador não será capaz de despertar em 10 segundos como desejado, sendo necessário, ao invés disso, despertar, por exemplo, de 5 em 5 ms durante 200 vezes.

Nossa rotina poderia, portanto, funcionar de acordo com o fluxograma da Fig. 5.30. O programa correspondente está na Fig. 5.31; caso desejássemos usá-lo, substituiríamos esta rotina no programa anterior (Fig. 5.29).

Programação do Relógio

Conforme discutido anteriormente a programação do relógio consiste em se carregar os registros de relógio como indicado:

O conteúdo de R4 e mais 6 bits de R5 indicarão a contagem, isto é, o número de pulsos ou sinais que deverão passar até que o relógio desperte. Os dois bits restantes de R5 indicarão o tipo de alarma (pulso ou onda quadrada) e se é desejado reinício automático.

Como desejamos despertar a cada 5 ms e o período do pulso de entrada é de 330 ns (relógio do processador), a contagem N desejada será 5 ms/330 ns ou 5.000 × 3 = 15.000. O tipo de alarma que nos interessa é um pulso no fim

da contagem e é desejável que a operação se repita indefinidamente sendo portanto os dois primeiros bits de R5 iguais a 1.

Para carregarmos esses registros, devemos notar que, pelo esquema de fiação do sistema mínimo (Fig. 5.25) os registros de relógio são as portas 12 e 13 (ou OC e OD em hexadecimal).

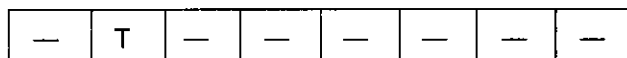
Após o carregamento desses registros ainda será necessário disparar o relógio, isto é, mandar que ele comece a funcionar. Isto é feito usando-se o registro de comando (porta 8) configurando-se os dois bits mais altos como 11. Os demais bits são irrelevantes nessa aplicação; poderemos zerá-los.

Finalmente, um outro aspecto interessante que deveremos discutir é como saber se o despertador está "tocando". Como já foi dito, no fim da contagem o relógio emitirá um sinal no pino de saída (TIMER OUT) que poderia avisar o processador que o tempo já transcorreu. Essa alternativa voltará a ser examinada quando tratarmos de interrupção. No momento podemos usar a alternativa mais simples de consultar o registro de estado onde o bit 6 reflete o estado do relógio:

```
                        ;****************************
                        ;           PROGRAMACAO DO RELOGIO
                        ;****************************
001D                            ORG     29
001D  C5                        PUSH    B        ;SALVA B E C
001E  01DC05                    LXI     B,1500   ;CARREGA CONTAGEM EM B E C
0021  78                        MOV     A,B      ;MOVE PARTE MAIS ALTA
0022  F6C0                      ORI     0C0H     ;ARMA BITS DE MODO
0024  D30D                      OUT     13       ;CARREGA NA PORTA 13
0026  79                        MOV     A,C      ;MOVE PARTE MAIS BAIXA
0027  D30C                      OUT     12       ;CARREGA NA PORTA 12
                        ;****************************
                        ;           DISPARAR O RELOGIO
                        ;****************************
0029  3EC0                      MVI     A,0C0H   ;ARMA BITS DE RELOGIO
002B  D308                      OUT     8        ;COLOCA NO REGISTRO COMANDO
                        ;****************************
                        ;           CONTAR O TEMPO
                        ;****************************
002D  01C800                    LXI     B,200    ;X=200
0030  DB08              AINDA:  IN      8        ;LER ESTADO
0032  E640                      ANI     40H      ;TESTA BIT DE ALARME E
0034  CA3000                    JZ      AINDA    ;DESVIA SE DESLIGADO
0037  05                        DCR     B        ;X=X-1
0038  C23000                    JNZ     AINDA    ;DESVIA SE X NAO FOR ZERO
                        ;****************************
                        ;           DESLIGAR O RELOGIO
                        ;****************************
003B  3E40                      MVI     A,40H    ;ZERAR ACUMULADOR
003D  D308                      OUT     8        ;COLOCAR NO REG. COMANDO
                        ;****************************
                        ;           FIM DA ROTINA
                        ;****************************
003F  C1                        POP     B        ;RESTAURA B E C
0040  C9                        RET              ;RETORNA
```

Fig. 5.31 – Uso do Despertador.

REGISTRO DE ESTADO

—	T	—	—	—	—	—	—

1 — CONTAGEM CHEGOU A ZERO
0 — CONTAGEM NÃO CHEGOU A ZERO

Entrada/Saída com Protocolo

Vimos que na 8156 as portas de entrada/saída podem funcionar no modo básico ou com protocolo. O uso de sinais de controle para protocolo, embora introduza alguma dificuldade traz também muita conveniência. Para ilustrar esta conveniência vamos supor que desejamos ler as informações contidas em uma fita de papel perfurada.

A fita de papel, como ilustra a Fig. 5.32 possui oito canais onde se pode ter uma informação. Uma perfuração corresponde a um "1" e a ausência da perfuração corresponde a um "0".

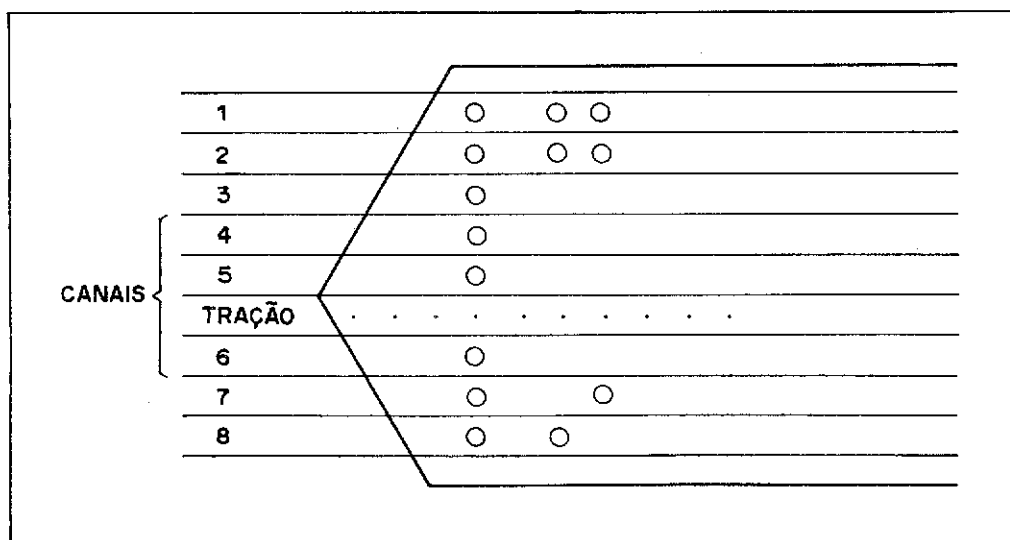



Fig. 5.32

Entre os canais 5 e 6 há um espaço destinado ao canal de tração. Neste lugar não há informação gravada, existe sempre um pequeno furo para que o mecanismo de tração do equipamento puxe a fita fazendo-a deslocar-se.

Para se ler a informação podemos pensar num equipamento como o da Fig. 5.33. Um motor faz com que a fita se desloque sob uma lâmpada. Quando um furo passa sob a lâmpada, a luz chega a uma fotocélula fazendo-a gerar um sinal lógico igual a 1. Existe uma fotocélula para cada canal de informação e uma sob o canal de tração.

EXEMPLO:
LEITORA DE FITA DE PAPEL
ENTRADA:
ON - LIGA O MOTOR
SAÍDA:
O0 - O7 - DADOS
SPK - TRAÇÃO
FOTOCÉLULAS
O0
O7
SPK
O0
A
O7
SPK
BF
8155
C
A INTR
ON
B
8085
RST 6.5

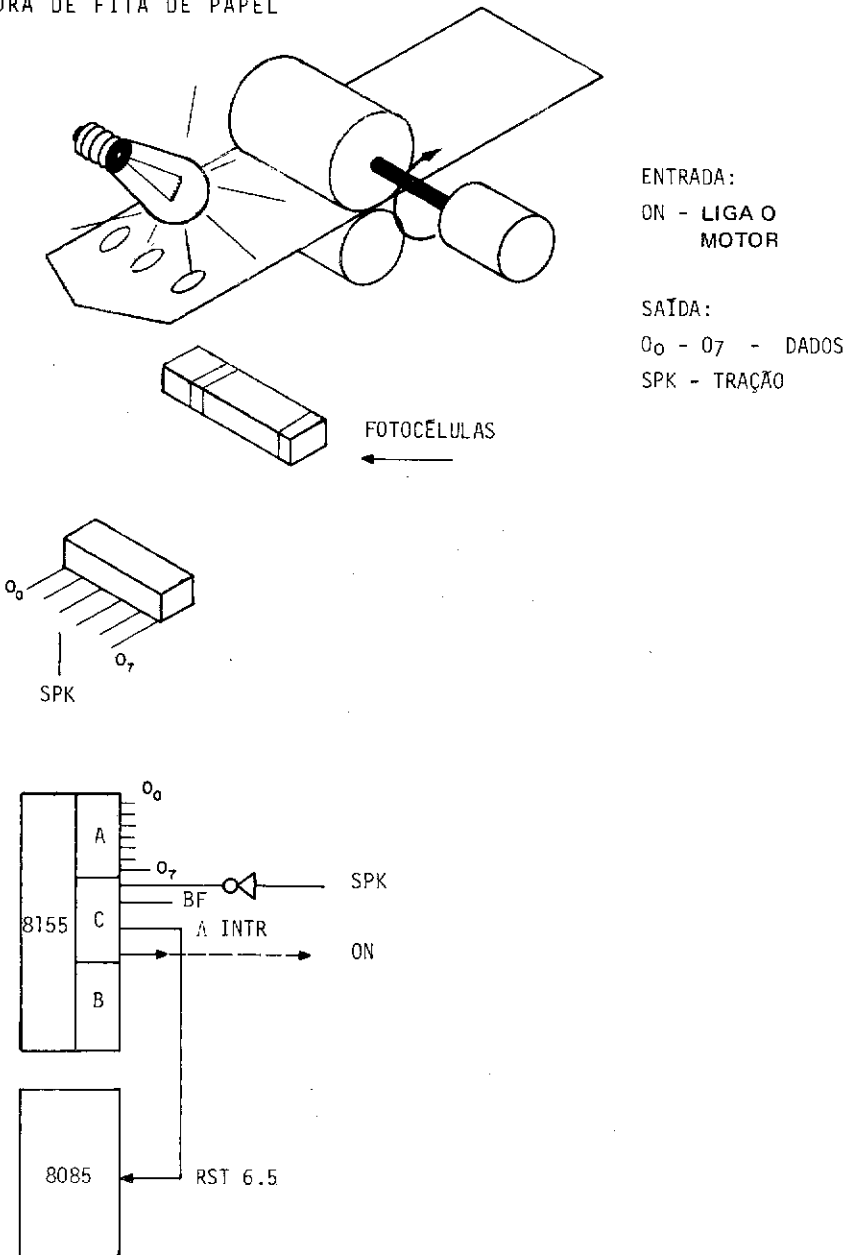



Fig. 5.33

A fotocélula sob o canal de tração serve para indicar o "momento oportuno" de se ler os dados, pois existe um intervalo entre um dado e outro onde nada está gravado. Como o furo de tração está centralizado com os de informação e tem um diâmetro menor, ele serve para sincronizar a leitura.

Adiante veremos a possibilidade de ligar a saída INTR na interrupção (RST 6.5) do processador.

Usando-se entrada/saída básica deveríamos proceder como indica o fluxograma da Fig. 5.34.

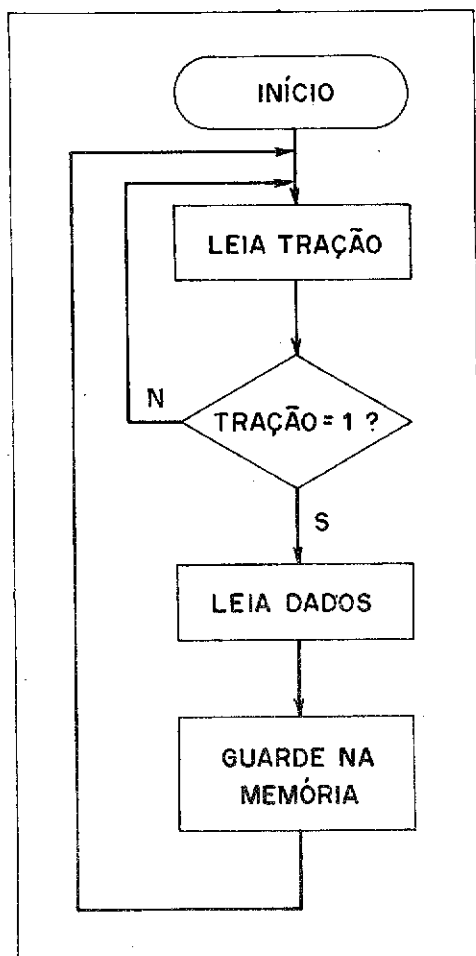



Fig. 5.34 — E/S básica.

Alguns inconvenientes deste método são:

1. Há um tempo crítico transcorrido entre a leitura da tração e dos dados, pois como a fita continua se movimentando, a leitura dos dados precisa ser feita enquanto os mesmos ainda estejam em posição.

2. Também há um tempo crítico para guardar os dados na memória; se houver algum atraso, o próximo furo de tração passará sem ser notado.

3. Sendo o computador muito rápido, é possível que se leia o mesmo dado mais de uma vez porque após guardá-lo na memória testamos o furo de tração que poderá ainda estar presente. Isto poderia ser contornado exigindo-se que a tração passasse por uma transição por zero.

Entrada com protocolo. A outra alternativa seria usar a entrada com protocolo disponível na 8156. Para isto o sinal mais importante é a "validação" pelo qual avisamos à porta de entrada que temos um dado disponível para leitura.

O próprio sinal de tração pode ser usado como validação sendo necessária uma inversão já que a validação é um sinal invertido. A validação nada mais é do que uma "habilitação de entrada" para o registro da porta da 8156. Com este sinal a informação é *copiada* e *armazenada* na porta. Assim, mesmo que o computador demore a buscar a informação, ela não será perdida até que chegue a próxima validação correspondendo ao dado seguinte.

Outra conveniência é que uma porta com protocolo pode gerar um sinal de pedido de interrupção que poderá ser usado para avisar ao processador que uma informação está disponível.

O fluxograma que seria seguido é apresentado na Fig. 5.35. O trecho de programa correspondente está na Fig. 5.36.

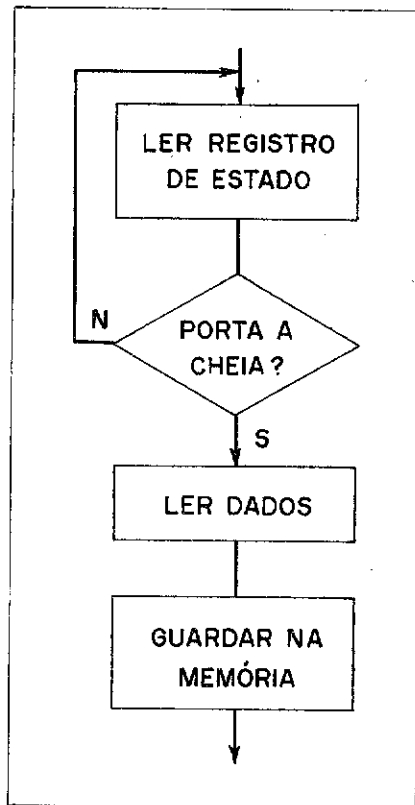



Fig. 5.35 — E/S com protocolo.

```
TESTE   IN     8      ;lê estado
        RRC           ;gira p/direita
                      ;coloca bit 0 no bit de
                      ;sinal
        JP TESTE      ;desvia se bit = 0
    PORTA A CONTÉM DADOS
        IN     9      ;lê porta A
    ARMAZENA NA MEMÓRIA
        ≡
```

Fig. 5.36 – Trecho de programa.

Uso de Interrupção

Sinal de pedestre. Ao invés de ligarmos o botão do pedestre a uma porta de entrada, poderíamos ligá-lo a uma das linhas de pedido de interrupção do 8085. Ao acionar o botão, o pedestre faria com que o computador desviasse para uma rotina especial; um novo fluxograma seria como mostra a Fig. 5.37.

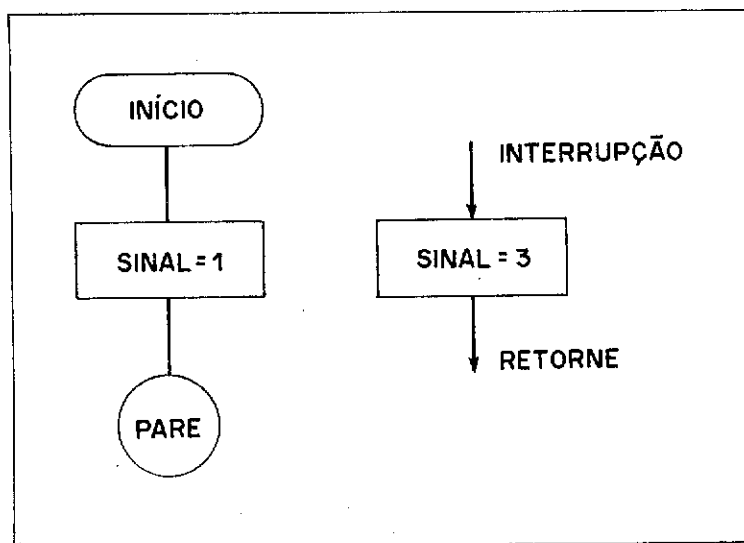



Fig. 5.37 – E/S com interrupção.

Neste caso, após inicializarmos o sinal em verde, pararíamos o computador (já que nada mais tem de ser feito). O computador deverá prosseguir após a intervenção do pedestre.

Quando o pedestre aciona o botão, o computador (que estaria executando a instrução PARE) desvia para a rotina de interrupção e, após executá-la, retorna à próxima instrução do programa.

Nesse caso, a rotina de interrupção é muito simples e serve apenas para retirar o computador da parada; tudo que é necessário é retornar ao programa e prosseguir de modo a controlar a seqüência de fechar o sinal.

Devemos apenas lembrar o seguinte:

a) a interrupção somente será atendida se a habilitação permitir (EI = 1);
b) quando uma interrupção é atendida, EI é feito zero até que o programa faça EI = 1.

É preciso, portanto, cuidar para que, no programa EI, esteja ligado quando for conveniente, isto é, enquanto o computador estiver parado. Para tanto basta que usemos uma instrução EI logo antes da parada.

Relógio. Outro exemplo útil de uso de interrupção seria para o controle do relógio. A linha de saída do relógio (TIMER OUT) poderia ser ligada a uma das interrupções do 8085.

Fita de papel. Mais relevante, a nosso ver, seria o uso de interrupção para a leitura de fita de papel exemplificada anteriormente. Suponhamos que se deseje ler e transferir para a memória uma certa quantidade de dados perfurados em fita de papel. É fato sabido e notório que os equipamentos de entrada/saída são bem mais lentos que a execução de instruções no computador. Por exemplo, uma leitora rápida de fita de papel que tivesse velocidade da ordem de 2.000 caracteres por segundo (2 kilobytes/segundo = 2KB/s) teria 1 caracter lido a cada 500 us, tempo em que o computador seria capaz de processar 200 instruções!

No entanto, este tempo seria perdido se o computador esperasse a chegada de cada caracter como foi sugerido. Usando interrupção o computador pode se dedicar a outro trabalho útil até que seja avisado sobre a chegada de um dado.

5.5. OUTRAS CONSIDERAÇÕES

Expansão do Sistema

O sistema mínimo apresentado pode ser usado numa grande variedade de aplicações. No entanto, há casos onde é necessário mais memória ou mesmo entrada/saída. A inclusão de mais circuitos da mesma família (8155/8156, 8355/8755) não deverá representar nenhuma dificuldade para o leitor (ver exercícios deste capítulo).

Em sistemas em que haja grandes requisitos de memória, poderá ser inconveniente o uso desses circuitos devido ao seu custo mais alto. As memórias comuns passarão a ser mais econômicas neste caso embora seja necessário descompartilhar as vias de endereços e dados.

A Fig. 5.38 ilustra um sistema com as vias descompartilhadas: um circuito adicional (8212) que nada mais é que um registro, é ligado às linhas AD_0 — AD_7 (compartilhadas por dados e endereços). Este circuito é comandado pela linha de controle ALE. A configuração dos bits de endereço presente em AD_0 — AD_7 é "fotografada" no interior desse registro e permanece inalterada quando, mais tarde, as linhas AD_0 — AD_7 são usadas para transferir dados.

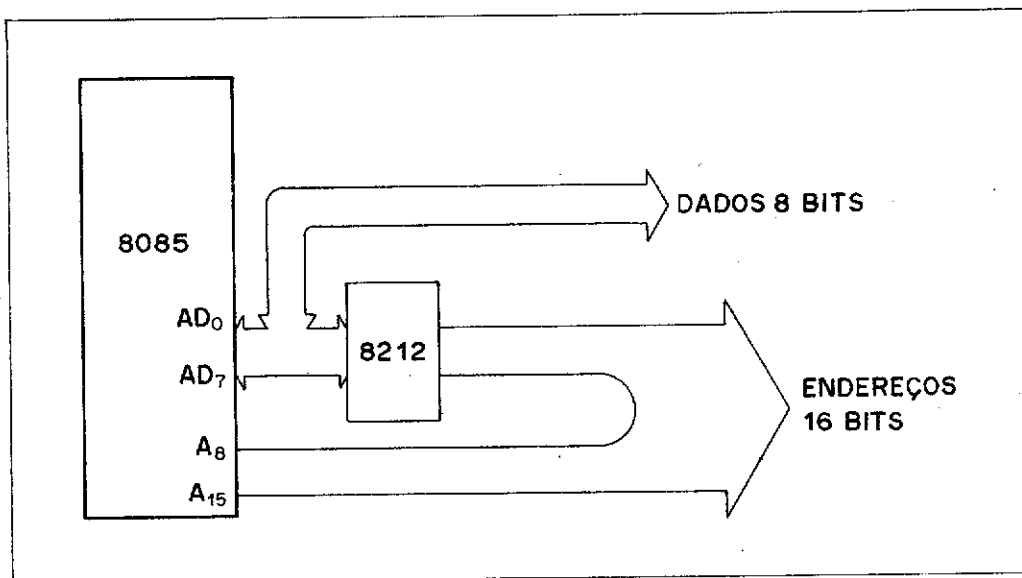



Fig. 5.38 – Descompartilhamento das vias.

Entrada/Saída Serial

Até o momento consideramos os dados de entrada/saída como um conjunto de bits que entram ou saem do computador simultaneamente, isto é, em paralelo. No caso mais comum tratam-se de 8 bits representando um caracter.

Neste tipo de transferência é necessário que haja um conjunto de, pelo menos, 8 fios (um para cada bit) ligando o dispositivo de entrada/saída à porta E/S do computador. Ocorre, no entanto, que em algumas situações, o dispositivo de E/S se encontra bastante afastado do computador e o custo da fiação pode pesar consideravelmente.

A idéia da transmissão em série consiste em enviar 1 bit de cada vez ao invés de 8 simultaneamente. Obviamente há um pouco mais de trabalho e a transferência é feita mais devagar do que em paralelo; no entanto, a economia é relevante.

Procedimento. Na transmissão paralela, o transmissor coloca a configuração de bits desejada ao longo dos 8 fios disponíveis que vão até o receptor. É comum também o uso de uma linha de validação que serve para avisar ao receptor que existem dados para serem lidos. (Isto é, exatamente, o que ocorre no exemplo da fita de papel.) O procedimento está ilustrado na Fig. 5.39.

Na transmissão serial, em que só se dispõe de 1 fio, o receptor examina os bits a serem transmitidos um a um fazendo com que a saída seja igual a 0 ou 1 conforme o valor deste bit. É necessário que tanto o transmissor quanto o receptor tenham um perfeito acordo sobre o tempo "t" de transmissão de cada bit; ou seja, se o tempo de 1 bit for 10 ms, o transmissor manterá a saída durante 10 ms, com o valor do bit que está sendo transmitido (ver Fig. 5.40).

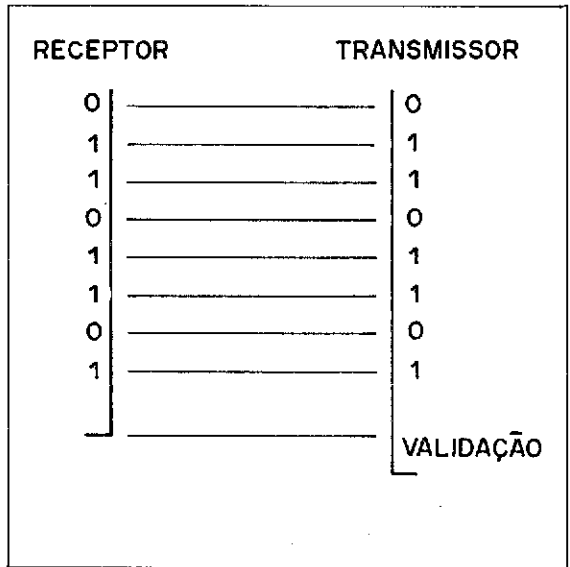



Fig. 5.39 — E/S Paralela.

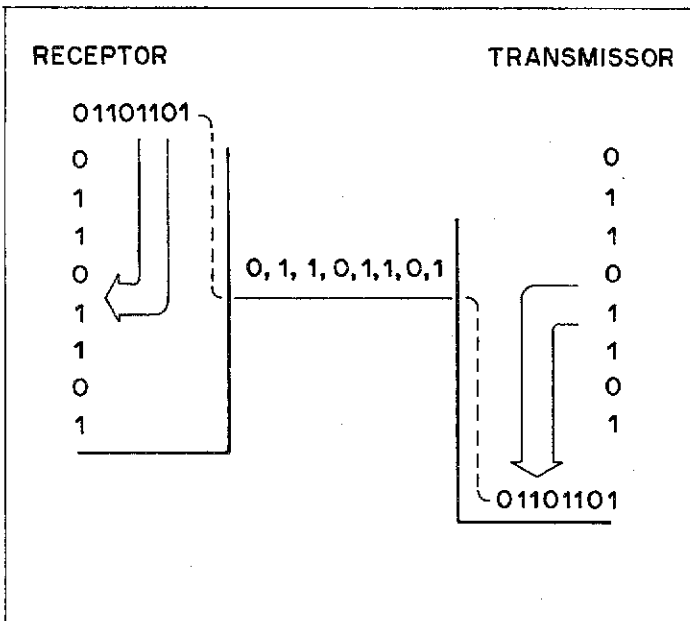



Fig. 5.40 — E/S em Série.

Velocidade de transmissão. Comumente, ao invés do tempo de 1 bit, ouviremos falar da velocidade de transmissão "V" dada em bits por segundo. O leitor pode facilmente deduzir que:

$$t = \frac{1}{V} \quad \text{ou} \quad V = \frac{1}{t}.$$

(Assim, se t = 10 ms = 0,01 s isto implicará numa velocidade de

$$V = \frac{1}{0.01} = 100 \text{ bps}).$$

Transmissão síncrona e assíncrona. Normalmente, na transmissão paralela um sinal de validação indica ao receptor o momento correto de se proceder à leitura dos dados. Na transmissão serial também é necessário avisar ao receptor o momento em que se inicia uma transmissão pois, mesmo sabendo o tempo de cada bit, o receptor poderia misturar parte dos bits finais de um caracter com parte dos bits iniciais do seguinte, bem como, durante um intervalo de transmissão continuar pensando em receber dados.

Claro que um sinal de validação usando um fio independente resolveria o problema. Poderíamos usar este fio para mandar apenas um sinal quando transmitíssemos o 1º bit de um caracter. Porém, uma solução evitando mais fios é claramente desejável.

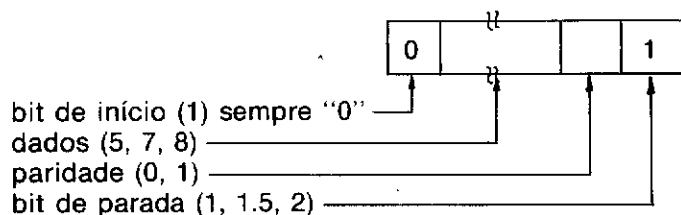
Formato assíncrono. A primeira solução, que é a mais comumente utilizada, consiste em sinalizar o início da transmissão para cada caracter transmitido da seguinte forma:

1. Inicialmente, e sempre que não houver transmissão, a linha é mantida no estado "1". (Assim, enquanto a linha é mantida em "1" o receptor sabe que não há transmissão.)

2. Para transmitir um caracter, o transmissor indica o início de transmissão deixando a linha no estado "0" durante o tempo de 1 bit, isto é, equivale a transmitir um bit de início (start bit). A seguir os demais bits do caracter.

3. Após transmitir o caracter o transmissor deixa a linha no estado "1" durante algum tempo; normalmente o tempo de 2 bits. Isto equivale a transmitir 2 bits "1" chamados bits de parada (stop bits).

Dessa forma, há uma oportunidade de que transmissor e receptor se sincronizem a cada caracter transmitido. A transmissão é chamada assíncrona porque a sincronização é refeita para cada caracter.

O formato do caracter transmitido/recebido é o seguinte:

Bit de início. Como vimos este bit é sempre zero e serve para indicar o início da transmissão (start bit).

Bits de dados. De acordo com o código utilizado podemos precisar de 5, 7 ou 8 bits. Comumente usa-se o código ASCII de 7 bits.

Paridade. Pode ou não ser usado. A paridade serve para detetar erros na transmissão e seu valor é calculado de forma a que o total de bits um "1" dos dados mais paridade seja ímpar (paridade ímpar) ou par (paridade par).

Bit de parada. Os bits de parada, que podem ser 1, 1 1/2 ou 2, servem apenas para dar tempo de ser feita uma nova sincronização. Note-se que esses bits se resumem em manter a linha em "1" durante o tempo de 1, 1 1/2 ou 2 bits.

(Isto explica o misterioso 1/2 bit.) Na verdade o número de bits de parada especifica o número mínimo de bits em "1" antes do próximo caracter pois, enquanto não houver transmissão, a linha permanecerá no estado "1".

Formato síncrono. A outra solução consiste em sincronizar transmissor e receptor para transmissão de um grande número de caracteres sem necessidade de ressincronização. Como visto, o formato assíncrono desperdiça, em geral, 3 bits (1 de início, 2 de fim) para a transmissão de um caracter de 7 bits e paridade; assim, para cada caracter temos pelo menos 3/8 do tempo desperdiçado.

Neste caso a sincronização é feita através de caracteres de sincronismo (SYNC), uma determinada configuração de bits destinada a este propósito. O transmissor envia um certo número de caracteres de sincronismo que permitem ao receptor sincronizar a recepção. Em seguida é transmitido um "cabeçalho" contendo informações sobre a transmissão que virá em seguida. Infelizmente há uma grande variedade possível em tais cabeçalhos dificultando as tentativas de padronização; assim, apesar de mais eficiente para transmissão de grandes números de caracteres a transmissão síncrona perde em universalidade para a assíncrona.

Padrões de transmissão: Elo de corrente e RS232C.

Apesar de que nos tenhamos utilizado do padrão TTL até o momento, este padrão não é adequado para transmissões distantes.

O elo de corrente consiste em se transmitir através de um fio duplo (ida e volta) uma corrente de 20 mA sendo o fluxo de corrente mantido ou não conforme se transmita um "1" ou um "0". Este padrão de transmissão é bem antigo, datando dos aparelhos de teletipo e é relativamente seguro.

Outro padrão muito mais recente obedece a norma RS232C da Electronics Industry Association — EIA usando níveis de tensão diferentes do TTL para transmissão/recepção e especificando, inclusive, um conector adequado a esta finalidade. Em geral usa-se o padrão RS-232-C quando há necessidade de ligação a um MODEM para transmissão por via telefônica.

Canais Full-Duplex e Half-Duplex. Usando-se um par de fios é possível mandar informação em ambos os sentidos, porém apenas um sentido deve ser usado de cada vez. Neste caso temos um canal "Half-Duplex". Em geral inicia-se uma tentativa de transmissão e um dos lados obtém o controle da mesma.

A comunicação "Full-Duplex" permite fluxo nos dois sentidos simultaneamente alocando um canal para transmissão e outro para recepção; um par de fios adicional é necessário.

Zilog Z80

Nesta seção discutiremos o microprocessador Z80 e seus componentes usuais. Os projetistas do Z80 não tinham uma preocupação de reduzir o número de circuitos necessários para uma configuração mínima. Ao invés disso, procuraram dotar este processador de características que facilitassem a programação e o projeto do sistema.

O Z80 também é compatível em linguagem de máquina com o 8080; isto quer dizer que um programa-objeto de 8080 pode ser executado sem qualquer alteração numa CPU de Z80. O Z80 tem instruções a mais que o 8080 aproveitando códigos de operação indefinidos.

O microprocessador Z80. Na Fig. 5.41 vemos os diagramas de pinagem e funcional do Z80. Note-se a não multiplexação dos sinais de endereços e dados,

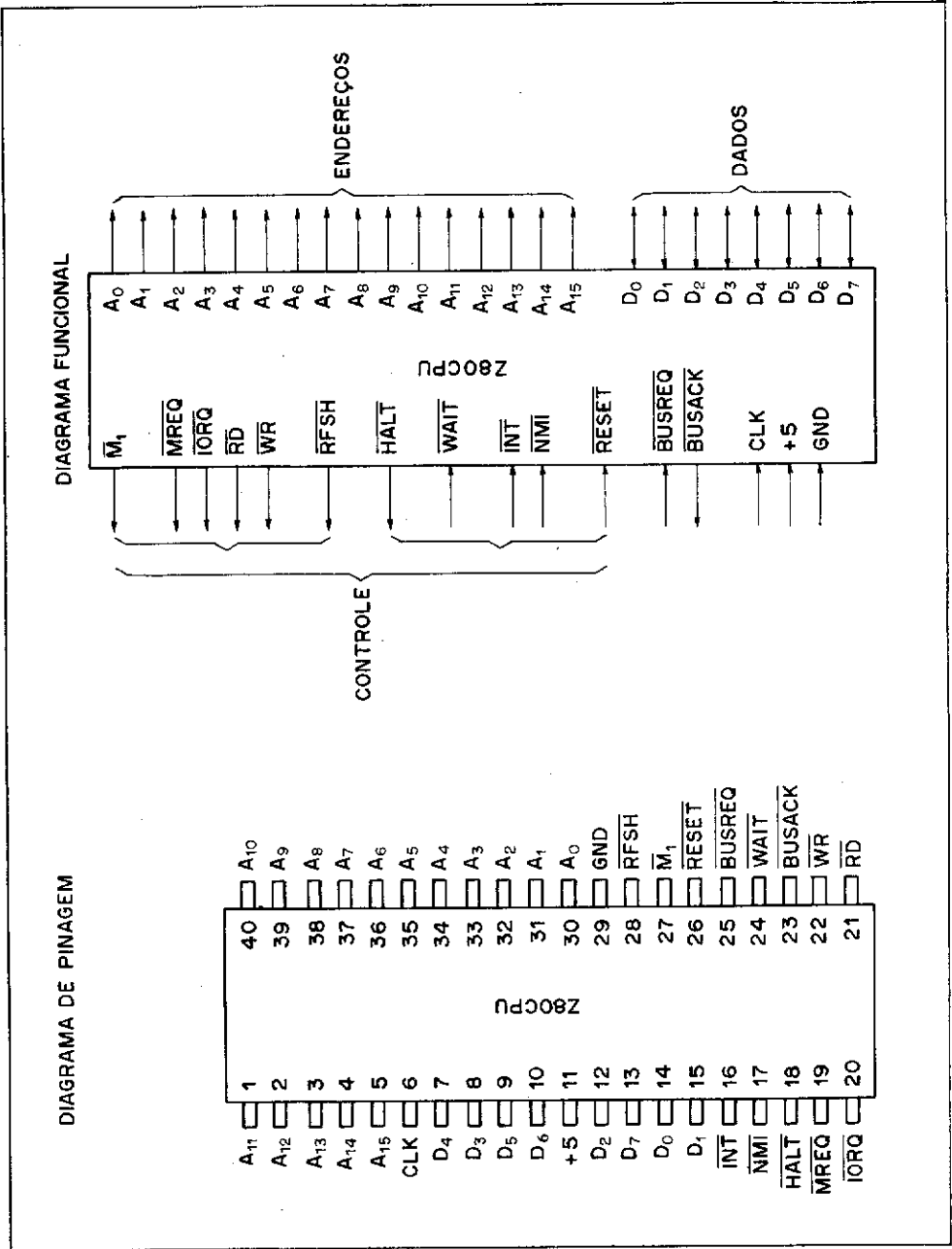



Fig. 5.41 — Microprocessador Z80.

um conceito que só apareceu mais tarde no 8085. Outro detalhe interessante é que o Z80 possui um pino de entrada do relógio (clock) isto porque *não há* relógio dentro da CPU sendo necessário um circuito externo para gerá-lo.

Via de endereços. Consiste dos sinais A_0 a A_{15} que são linhas de saída de três estados que contêm o endereço da memória ou da porta de E/S referenciada pelo processador. Os endereços de E/S, que só ocupam 8 bits são enviados apenas pelas linhas A_0 a A_7. (Não são duplicados como no 8085!)

Via de dados. Consiste de 8 sinais D_0 a D_7 que são linhas bidirecionais de 3 estados que transmitem dados entre os diversos componentes do sistema.

Via de controle

$\overline{RD}$ — (Leitura) — $\overline{RD} = 0$
indica que o processador aguarda dados de uma leitura de memória ou de porta E/S (3 estados).

$\overline{WR}$ — (Escrita) — $\overline{WR} = 0$
indica que a via de dados contém dados prontos para serem armazenados na memória ou porta E/S (3 estados).

$\overline{MREQ}$ — (Memória) — MREQ = 0
indica que as linhas de endereço contêm um endereço válido para uma leitura ou escrita na memória (3 estados).

$\overline{IORQ}$ — (entrada/saída) — IORQ = 0
indica que existe um endereço de entrada/saída válido para uma operação (3 estados).

$\overline{WAIT}$ — (Espera)
é um sinal que entra no processador para sincronizá-lo com memórias lentas (como o READY do 8085) enquanto WAIT = 0 o processador suspende sua operação.

$\overline{RESET}$ — (Inicialização)
é um sinal que entra no processador para indicar que uma inicialização é necessária. Dentre outras coisas o apontador de programa (PC) é zerado. O sinal RESET deve permanecer ativo pelo menos durante três ciclos do relógio.

$\overline{INT}$ — pedido de interrupção
sinal que entra no processador usado para indicar que um dispositivo externo está solicitando interrupção.

$\overline{RFSH}$ — Refrescar memória
sinal de saída do processador que é ativado simultaneamente com MREQ indicando que os bits A_0 — A_6 contêm um endereço que deve ser usado para refrescar memórias dinâmicas.

$\overline{HALT}$ — Parado
sinal de saída que indica que o processador encontra-se parado aguardando uma interrupção para prosseguir. Mesmo neste estado o processador continua gerando sinais para refrescar a memória.

$\overline{NMI}$ — Interrupção forçada
sinal de entrada indicando que um dispositivo externo solicita

interrupção. Este tipo de interrupção não pode ser ignorado pelo processador que será portanto forçado a obedecê-la. NMI tem mais prioridade que INT e força uma instrução restart para o endereço hexadecimal 0066.

$\overline{M1}$ — Início de ciclo (sinal de saída)
$\overline{M1}$ junto com $\overline{MREQ}$ indica que a CPU está executando busca (fetch) de instrução.
$\overline{M1}$ junto com $\overline{IORQ}$ indica que a CPU vai atender a um pedido de interrupção (*Acknowledge*).

$\overline{BUSREQ}$ — Pedido de vias
sinal de entrada usado por dispositivos externos para utilização das vias com a finalidade de transferir dados. O pedido é sempre atendido logo que a CPU terminar o ciclo atual.

$\overline{BUSACK}$ — Via livre
sinal de saída do processador que indica que as vias estão livres em resposta a um pedido $\overline{BUSREQ}$. As vias de endereços, dados e os sinais $\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$, $\overline{WR}$ são postos em alta impedância pelo processador para que sejam dirigidos por um dispositivo externo. (Maiores detalhes na seção ENTRADA/SAÍDA com DMA.)

Outros Circuitos de Interesse

Nesta seção descrevemos brevemente alguns circuitos integrados de interesse em sistemas microprocessadores. Outros circuitos bem como maiores detalhes sobre estes mesmos podem ser encontrados no manual MCS-85 USER's MANUAL da Intel Corp.

8212 — REGISTRO DE 8 BITS

O circuito 8212 é um registro de 8 bits que pode ter, dentre outras aplicações, a de porta de entrada, porta de saída e descompartilhador de vias para o 8085.

DIAGRAMA DE PINAGEM

Sinal	Pino	8212	Pino	Sinal
$\overline{DS_1}$	1		24	V_{CC}
MD	2		23	$\overline{INT}$
DI_1	3		22	DI_8
DO_1	4		21	DO_8
DI_2	5		20	DI_7
DO_2	6		19	DO_7
DI_3	7		18	DI_6
DO_3	8		17	DO_6
DI_4	9		16	DI_5
DO_4	10		15	DO_5
STB	11		14	$\overline{CLR}$
GND	12		13	DS_2

DI_1 — DI_8	entradas
DO_1 — DO_8	saídas
$\overline{DS_1}$ — DS_2	seleção (habilitação da pastilha)
MD —	modo
STB —	validação (habilitação dos dados)
$\overline{INT}$ —	interrupção
$\overline{CLR}$ —	limpeza

Fig. 5.42A – Pinagem do 8212.

Fig. 5.42B — 8212 como porta de entrada.

Fig. 5.42C — 8212 como porta de saída.

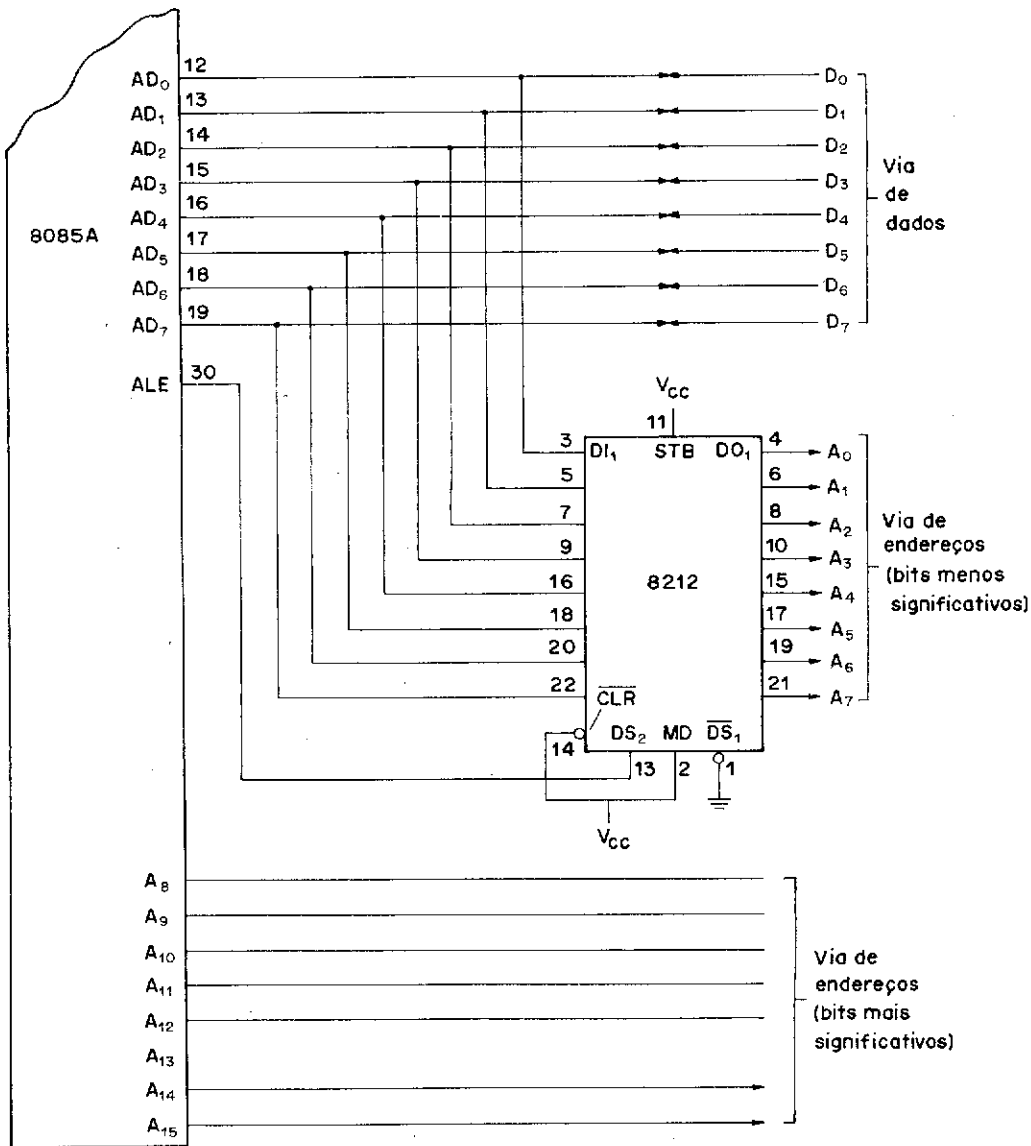



Fig. 5.42D – 8212 como descompartilhador de vias.

8205 — DECODIFICADOR RAPIDO 1 DE 8

Descrição. O 8205 é um decodificador binário 1 de 8. Isto significa que apenas uma de suas oito saídas será ativada, conforme o estado das três entradas. Desta forma, se as entradas A_0, A_1 e A_2 formarem a configuração 110 respectivamente, quando a pastilha for liberada (através da porta de "enable"), apenas a saída 03 será ativada indo para 0.

Para sincronizar a operação do decodificador com o resto do circuito, dispomos no 8205 de três entradas de habilitação, duas ativas em 0 e uma ativa em 1. A pastilha só será liberada se as três entradas estiverem ativas.

Note-se que as saídas são invertidas, isto é, as saídas desligadas estão em 1 e a ligada está em 0.

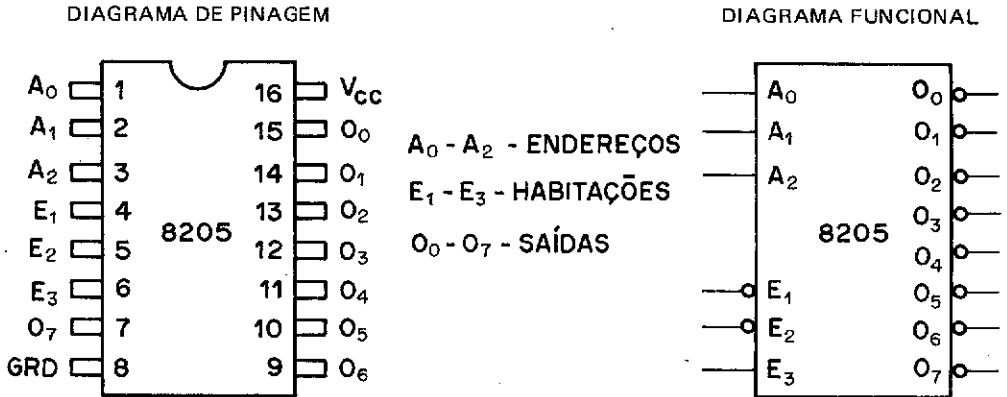



Fig. 5.43 – Decodificador 8205.

2716 — 2K × 8UV EPROM

A 2716 é uma memória ROM apagável por ultravioleta e programável eletricamente. Cada pastilha pode armazenar 16K bits, organizados em 2K bytes.

1. *Apagamento.* A 2716 pode ser apagada através da exposição à luz ultravioleta. O tempo necessário para o apagamento depende da potência da lâmpada. Uma lâmpada de 12.000 W/cm² apagaria em aproximadamente 15 a 20 minutos.

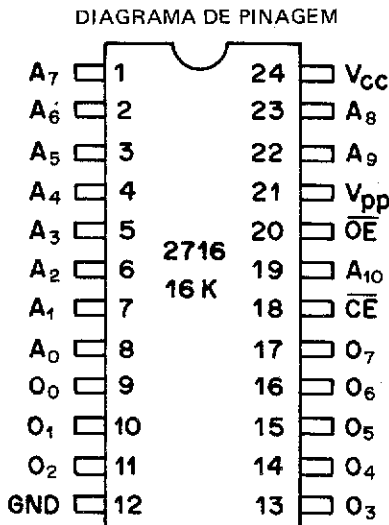



Fig. 5.44 – Memória EPROM 2716.

2. *Gravação.* Após o apagamento, todos os bits da memória estão em 1. O processo de gravação consiste então em "zerar" os bits necessários. Embora apenas os "zeros" sejam gravados, podemos ter tanto "1", como "0" presentes na barra de dados durante a gravação. Durante a gravação, o pino de V_{pp} é mantido em 25V. Após carregarmos o endereço e os dados nas barras, um único pulso de 5V no pino $\overline{CE}$ programa o dado até novo apagamento.

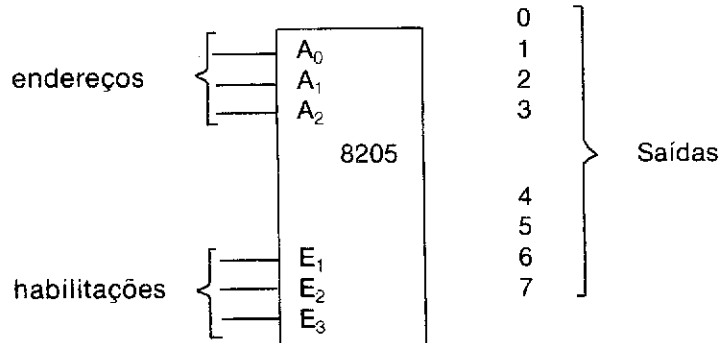
3. *Leitura.* Durante a operação normal, a 2716 é habilitada por 2 sinais: $\overline{CE}$ e $\overline{OE}$. Se o $\overline{CE}$ está inativo, a pastilha entra no estado "Standby", dissipando apenas 132 mW, as saídas vão para alta impedância, independendo do estado da entrada $\overline{OE}$. A entrada $\overline{OE}$ está inativa e a saída vai para alta impedância, independendo do CE.

4. *Tabela de Operação*

MODO \ PINO	$\overline{CE}$/PGM (18)	$\overline{OE}$ (20)	V_{pp} (21)	SAÍDAS
Leitura	0	0	+ 5V	Valor da saída
Inoperante	1	qualquer	+ 5V	Alta impedância
Programação	0 para 1	1	+25V	Valor de entrada
Verificação	0	0	+25V	Valor de saída
Inibição	0	1	+25V	Alta impedância

5.6. EXERCÍCIOS

1. Como deveríamos efetuar as ligações em um sistema mínimo de tal forma que o endereço $301A_{16}$ estivesse contido na RAM? Quais seriam os endereços das portas A e B da RAM neste caso?

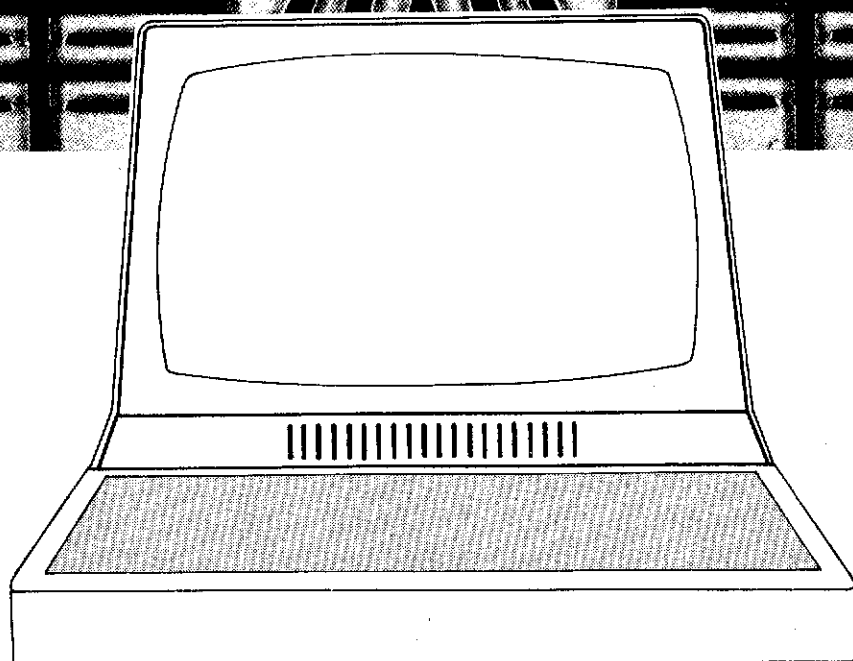
2. Suponha que desejamos projetar um sistema 8085 para usar 2K de ROM (8755) e 2K de RAM (8155 ou 8156). Dispõe-se de um decodificador como mostra a figura abaixo. Note que as saídas do decodificador são invertidas (a saída habilitada está em zero, as demais em 1) e que o mesmo tem 3 habilitações sendo duas invertidas, isto é, o decodificador funciona quando $\overline{E}_1 = 0$, $\overline{E}_2 = 0$ e $E_3 = 1$.

a) Projete o sistema.

b) Responda também qual será o endereço do registro de comando em cada uma das RAM's.

3. No exercício 2 é impossível utilizar as portas de E/S das 8155 se o endereço das memórias RAM for contíguo aos 2K da ROM. Explique porquê.

4. Se construirmos uma leitora de fita de papel usando interrupção, qual a velocidade máxima desta leitora em termos de caracteres por segundo?

Sugestão: Calcular o tempo necessário, no pior caso, para que a rotina de interrupção processe cada caracter recebido. (Isso envolve, para cálculo exato, fazer a rotina.)

5. A pastilha 2716 é uma memória do tipo EPROM. Projete um circuito copiador de EPROM 2716. Tal circuito deve ler cada byte de uma 2716 e gravá-lo em outra.

CONCLUSÕES (POSFÁCIO)

Vimos, nos Caps. 2 e 5, uma introdução à arquitetura de sistemas baseados em microprocessadores. Esperamos que o leitor agora disponha de elementos suficientes para conhecer, intimamente, o funcionamento de microcomputadores, em particular aqueles baseados em processadores da família 8080, bem como, por extrapolação simples, o funcionamento de computadores de maior porte.

Obviamente o assunto não está encerrado e o leitor interessado ainda terá muito material pela frente. É nossa firme convicção que a base de conhecimentos adquirida irá permitir um estudo aprofundado sem grandes dificuldades.

Com relação ao estudo de programação, visto nos Caps. 3 e 4, acreditamos ter coberto de forma completa o assunto podendo oferecer ainda ao leitor material para referência.

Chamamos a atenção do leitor para que o microprocessador seja visto num espectro muito mais amplo do que o processamento de dados. Os microprocessadores são usados hoje em controladores de tráfego, instrumentos médico-cirúrgicos, instrumentos de laboratório, automóveis, aviões, máquinas de costura e brinquedos.

O uso do microprocessador nesta situação requer, do projetista, um bom conhecimento do "Sistema" que o envolve. Freqüentemente o sistema envolve a manipulação de variáveis analógicas tais como temperatura, pressão etc. Neste caso há a necessidade de se converter o valor analógico para um valor digital. Isto é, um número que exprime sua medida. O processador trabalha apenas com os valores digitais dando também, como saída, resultados digitais que serão, se necessário, convertidos para valores analógicos.

E agora?

Apesar de ter provido material suficiente para o perfeito entendimento de como funcionam os computadores, não será possível, usando apenas nosso texto, realizar projetos de equipamentos. Propositalmente foram omitidos vários detalhes práticos que tornariam a iniciação ao assunto extremamente penosa; é nosso julgamento que tais detalhes devam ser incorporados aos conhecimentos do leitor em processo separado.

Assim, os leitores que desejarem prosseguir nesta direção, já se encontram, praticamente, em condições de acompanhar o manual do fabricante.

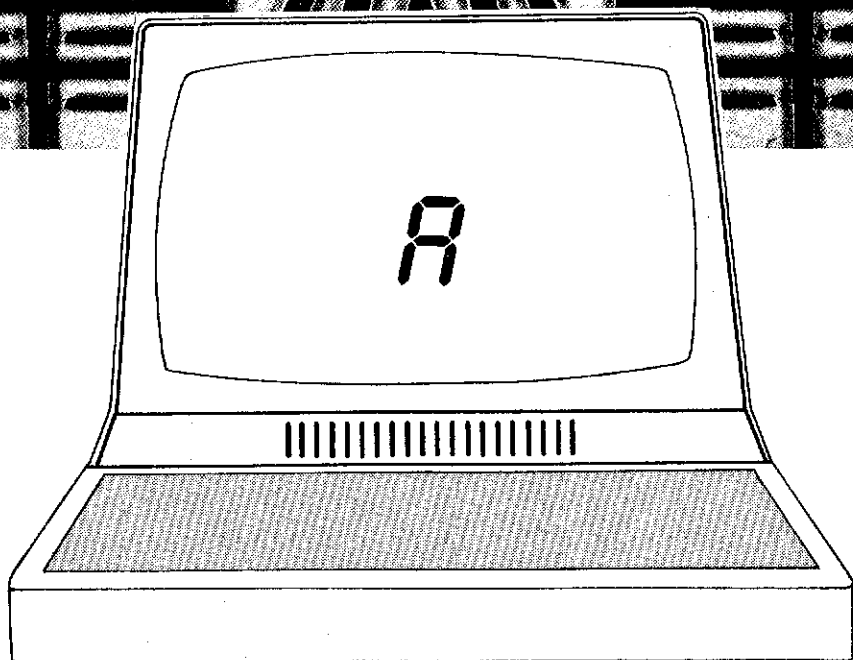
Relativamente à programação, advertimos ao leitor iniciante para não subestimar a complexidade do trabalho. A programação de sistemas complexos requer muita experiência do programador para resultar em um funcionamento correto e eficiente. Aqueles que desejarem ampliar seus conhecimentos especificamente em microcomputadores usando CP/M poderão se referenciar aos manuais deste Sistema.

Para desenvolver programação complexa o leitor necessitará ampliar seus conhecimentos de forma mais ampla sobre programação e sistemas operacionais, o que não é possível de ser feito através de manuais de fabricantes, mas sim usando livros ou freqüentando cursos especializados.

A tecnologia ligada aos computadores é muito ativa e evolui muito rapidamente. Novos produtos são lançados periodicamente e, em muitos casos, outros são retirados de circulação tornando "obsoletos" os conhecimentos a seu respeito.

Não devemos, portanto, nos iludir com a utilidade de conhecimentos específicos. Apesar de que os micros da família 8080 ainda tenham um bom tempo

de vida útil, outros produtos já começam a entrar no mercado. Por isso, procuramos, sempre que possível, dar mais ênfase aos conceitos, que permanecerão válidos por várias gerações de computadores, do que a itens específicos, que são válidos apenas para um sistema particular.

Microcomputadores de 16 bits, cujo desempenho e gama de aplicações é bem superior aos dos que examinamos, constituirão um mercado à parte coexistindo com os micros de 8 bits durante um bom tempo. Nesses sistemas, os conceitos são inteiramente os mesmos e o leitor também estará em condições para acompanhar a descrição de funcionamento dada pelo manual do fabricante.

SUMÁRIO DE INSTRUÇÕES 8080/8085

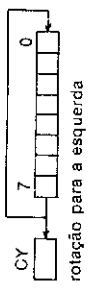
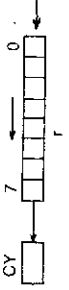
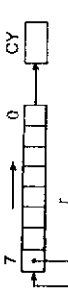
TIPO	MNEMÔNICO	OPERANDO	CÓDIGO-OBJETO	PERÍODOS		TAMANHO	INDICADORES — FLAGS					DESCRIÇÃO
				8080	8085		CY	Z	S	P	Ac	
TRANSFERÊNCIA DE DADOS	MOV	r_1, r_2	01 r_1 r_2	5	4	1						$r_1 \leftarrow r_2$
	MOV	r,M	01 r 110	7	7	1						$r \leftarrow (HL)$
	MOV	M,r	01 110 r	7	7	1						$(HL) \leftarrow r$
	MVI	r,e	00 r 110 e	7	7	2						$r \leftarrow e$
	MVI	M,e	36	10	10	2						$(HL) \leftarrow e$
	LXI	B,e16	01 e_1 e_2	10	10	3						$B \leftarrow e_2$, $C \leftarrow e_1$
	LXI	D,e16	11 e_1 e_2	10	10	3						$D \leftarrow e_2$, $E \leftarrow e_1$
	LXI	H,e16	21 e_1 e_2	10	10	3						$H \leftarrow e_2$, $L \leftarrow e_1$
	LXI	SP,e16	31 e_1 e_2	10	10	3						$SP_A \leftarrow e_2$, $SP_B \leftarrow e_1$
	LDA	end	3A ← end →	13	13	3						A ← (end)
	STA	end	32 ← end →	13	13	3						(end) ← A
	LHLD	end	2A ← end →	16	16	3						H ← (end+1), L ← (end)
	SHLD	end	22 ← end →	16	16	3						(end+1) ← H, (end) ← L
	LDAX	B	0A	7	7	1						A ← (BC)
	LDAX	D	1A	7	7	1						A ← (DE)
	STAX	B	02	7	7	1						(BC) ← A
	STAX	D	12	7	7	1						(DE) ← A
	XCHG		EB	4	4	1						H ← → D, L ← → E
ARITMÉTICAS	ADD	r	10000 r	4	4	1	X	X	X	X	X	A ← A + r
	ADD	M	86	7	7	1	X	X	X	X	X	A ← A + (HL)
	ADI	e	C6 e	7	7	2	X	X	X	X	X	A ← A + e
	ADC	r	10001 r	4	4	1	X	X	X	X	X	A ← A + r + CY
	ADC	M	8E	7	7	1	X	X	X	X	X	A ← A + (HL) + CY
	ACI	e	CE e	7	7	2	X	X	X	X	X	A ← A + e + CY
	SUB	r	10010 r	4	4	1	X	X	X	X	X	A ← A − r
	SUB	M	96	7	7	1	X	X	X	X	X	A ← A − (HL)
	SUI	e	D6 e	7	7	2	X	X	X	X	X	A ← A − e
	SBB	r	10011 r	4	4	1	X	X	X	X	X	A ← A − r − CY
	SBB	M	9E	7	7	1	X	X	X	X	X	A ← A − (HL) − CY
	SBI	e	DE e	7	7	2	X	X	X	X	X	A ← A − e − CY
	INR	r	00r100	5	4	1		X	X	X	X	r ← r + 1
	INR	M	34	10	10	1		X	X	X	X	(HL) ← (HL) + 1
	DCR	r	00r101	5	4	1		X	X	X	X	r ← r − 1
	DCR	M	35	10	10	1		X	X	X	X	(HL) ← (HL) − 1
	INX	B	03	5	6	1						BC ← BC + 1
	INX	D	13	5	6	1						DE ← DE + 1
	INX	H	23	5	6	1						HL ← HL + 1
	INX	SP	33	5	6	1						SP ← SP + 1

TIPO	MNEMÔNICO	OPERANDO	CÓDIGO-OBJETO		PERÍODOS		TAMANHO	INDICADORES – FLAGS					DESCRIÇÃO
					8080	8085		CY	Z	S	P	Ac	
ARITMÉTICAS	DCX	B	0B		5	6	1						BC ← BC − 1
	DCX	D	1B		5	6	1						DE ← DE − 1
	DCX	H	2B		5	6	1						HL ← HL − 1
	DCX	SP	3B		5	6	1						SP ← SP − 1
	DAD	B	09		10	10	1	X					HL ← HL + BC
	DAD	D	19		10	10	1	X					HL ← HL + DE
	DAD	H	29		10	10	1	X					HL ← HL + HL
	DAD	SP	39		10	10	1	X					HL ← HL + SP
	DAA		27		4	4	1	X	X	X	X	X	O conteúdo do acumulador é ajustado de forma a a se obter 2 dígitos em BCD. 1. Se os 4 bits menos significativos do acumulador forem maiores que 9 e o Ac = 1, 6 é somado ao acumulador 2. Se os 4 bits mais significativos do acumulador são agora maiores que 9, ou CY = 1, 6 é somado os 4 bits mais significativos do acumulador
LÓGICAS	ANA	r	10100 r		4	4	1	0	X	X	X	1	A ← A ∧ r
	ANA	M	A6		7	7	1	0	X	X	X	1	A ← A ∧ (HL)
	ANI	e	E6	e	7	7	2	0	X	X	X	1	A ← A ∧ e
	XRA	r	10101 r		4	4	1	0	X	X	X	0	A ← A ∀ r
	XRA	M	AE		7	7	1	0	X	X	X	0	A ← A ∀ (HL)
	XRI	e	EE	e	7	7	2	0	X	X	X	0	A ← A ∀ e
	ORA	r	10110 r		5	4	1	0	X	X	X	0	A ← A ∨ r
	ORA	M	B6		7	7	1	0	X	X	X	0	A ← A ∨ (HL)
	ORI	e	F6	e	7	7	2	0	X	X	X	0	A ← A ∨ e
	CMP	r	10111 r		4	4	1	X	X	X	X	X	A−r, compara o acumulador com o registro r
	CMP	M	BE		7	7	1	X	X	X	X	X	A−(HL), compara o acumulador com a posição de memória apontada por HL
	CPI	e	FE		7	7	2	X	X	X	X	X	A−e compara o acumulador com o segundo byte da instrução
	RLC		07		4	4	1	X					CY 7 0 rotação para a esquerda

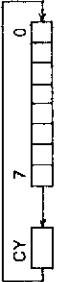
TIPO	MNEMÔNICO	OPERANDO	CÓDIGO-OBJETO		PERÍODOS		TAMANHO	INDICADORES – FLAGS					DESCRIÇÃO
					8080	8085		CY	Z	S	P	Ac	
LÓGICAS	RRC		OF		4	4	1	X					CY 7 0; rotação para a direita
	RAL		17		4	4	1	X					CY 7 0; rotação para a esquerda, através do **VAI UM**
	RAR		1F		4	4	1	X					CY 7 0; rotação para a direita, através do **VAI UM**
	CMA		2F		4	4	1						$A \leftarrow \overline{A}$, complemento a um **VAI UM**
	CMC		3F		4	4	1	X					$CY \leftarrow \overline{CY}$, complementa o **VAI UM**
	STC		37		4	4	1	X					
DESVIO	JMP	end	C3	← end →	10	10	3						PC – end, desvio incondicional
	JNZ	end	C2	← end →	10	7/10	3						Se Z = 0,
	JZ	end	CA	← end →	10	7/10	3						Se Z = 1,
	JNC	end	D2	← end →	10	7/10	3						Se CY = 0,
	JC	end	DA	← end →	10	7/10	3						Se CY = 1, [PC ← end]
	JPO	end	E2	← end →	10	7/10	3						Se P = 0,
	JPE	end	EA	← end →	10	7/10	3						Se P = 1,
	JP	end	F2	← end →	10	7/10	3						Se S = 0,
	JM	end	FA	← end →	10	7/10	3						Se S = 1
	CALL	end	CE	← end →	17	18	3						$(SP\text{-}1) \leftarrow PC_A$; $(SP\text{-}2) \leftarrow PC_B$; $SP \rightleftarrows SP\text{-}2$; $PC \leftarrow end$
	CNZ	end	C4	← end →	11/17	9/18	3						Se Z = 0,
	CZ	end	CC	← end →	11/17	9/18	3						Se Z = 1,
	CNC	end	D4	← end →	11/17	9/18	3						Se CY = 0,
	CC	end	DC	← end →	11/17	9/18	3						Se CY = 1, $[(SP\text{-}1) \leftarrow PC_A$; $(SP\text{-}2) \leftarrow PC_B$;
	CPO	end	E4	← end →	11/17	9/18	3						Se P = 0, $SP \leftarrow SP\text{-}2$; $PC \leftarrow end]$
	CPE	end	EC	← end →	11/17	9/18	3						Se P = 1,
	CP	end	F4	← end →	11/17	9/18	3						Se S = 0,
	CM	end	FC	← end →	11/17	9/18	3						Se S = 1
	RET		C9		10	10	1						$PC_B \leftarrow (SP)$; $PC_A \leftarrow (SP+1)$; $SP \leftarrow SP+2$

SUMÁRIO DE INSTRUÇÕES Z80

TIPO	MNEMÔNICO	OPERANDO	CÓDIGO-OBJETIVO	PERÍODOS	TAMANHO	INDICADORES – FLAGS						DESCRIÇÃO
						CY	Z	S	P/O	Ac	N	
TRANSFERÊNCIA DE DADOS	LD	BC,e16	01 ← e16 →	10	3							BC ← e16
	LD	DE,e16	11 ← e16 →	10	3							DE ← e16
	LD	HL,e16	21 ← e16 →	10	3							HL ← e16
	LD	SP,e16	31 ← e16 →	10	3							SP ← e16
	LD	IX,e16	DD 21 ← e16 →	14	4							IX ← e16
	LD	IY,e16	FD 21 ← e16 →	14	4							IY ← e16
	LD	HL,(end)	2A ← end →	16	3							H ← (end + 1) e L ← (end)
	LD	BC,(end)	ED 4B ← end →	20	4							B ← (end + 1) e C ← (end)
	LD	DE,(end)	ED 5B ← end →	20	4							D ← (end + 1) e E ← (end)
	LD	HL,(end)	ED 6B ← end →	20	4							H ← (end + 1) e L ← (end)
	LD	SP,(end)	ED 7B ← end →	20	4							SP_A ← (end + 1) e SP_B ← (end)
	LD	IX,(end)	DD 2A ← end →	20	4							IX_A ← (end + 1) e IX_B ← (end)
	LD	IY,(end)	FD 2A ← end →	20	4							IY_A ← (end + 1) e IY_B ← (end)
	LD	(end),HL	22 ← end →	16	3							(end + 1) ← H e (end) ← L
	LD	(end),BC	ED 43 ← end →	20	4							(end + 1) ← B e (end) ← C
	LD	(end),DE	ED 53 ← end →	20	4							(end + 1) ← D e (end) ← E
	LD	(end),HL	ED 63 ← end →	20	4							(end + 1) ← H e (end) ← L
	LD	(end),SP	ED 73 ← end →	20	4							(end + 1) ← SP_A e (end) ← SP_B
	LD	(end),IX	DD 22 ← end →	20	4							(end + 1) ← IX_A e (end) ← IX_B
	LD	(end),IY	FD 22 ← end →	20	4							(end + 1) ← IY_A e (end) ← IY_B
	LD	SP,HL	F9	6	1							SP ← HL
	LD	SP,IX	DD F9	10	2							SP ← IX
	LD	SP, IY	FD F9	10	2							SP ← IY
	LD	r_1,r_2	01 r_1 r_2	4	1							r_1 ← r_2
	LD	r,(HL)	01 r 110	7	1							r ← (HL)
	LD	r,(IX+e)	DD 01 r 110 e	19	3							r ← (IX + e)
	LD	r,(IY+e)	FD 01 r 110 e	19	3							r ← (IY + e)
	LD	(HL),r	01110 r	7	1							(HL) ← r
	LD	(IX+e),r	DD 01110 r e	19	3							(IX+e) ← r
	LD	(IY+e),r	FD 01110 r e	19	3							(IY+e) ← r
	LD	r,e	00 r 110 e	7	2							r ← e
	LD	(HL),e	36 e	10	2							(HL) ← e
	LD	$(IX+e_1),e_2$	DD 36 e_1 e_2	19	4							$(IX+e_1)$ ← e_2
	LD	$(IY+e_1),e_2$	FD 36 e_1 e_2	19	4							$(IY+e_1)$ ← e_2
	LD	A,(BC)	0A	7	1							A ← (BC)
	LD	A,(DE)	1A	7	1							A ← (DE)
	LD	A,(end)	3A	13	3							A ← (end)
	LD	(BC), A	02	7	1							(BC) ← A
	LD	(DE), A	12	7	1							(DE) ← A
	LD	(end), A	32 ← end →	13	3							(end) ← A

TIPO	MNEMÔNICO	OPERANDO	CÓDIGO-OBJETIVO			PERÍODOS	TAMANHO	INDICADORES – FLAGS						DESCRIÇÃO
								CY	Z	S	P/O	Ac	N	
TRANSFERÊNCIA DE DADOS	LD	A, I	ED	57		9	2		X	X	1	IFF*	0	A ← I transfere para o acumulador o conteúdo do registro Vetor de Interrupção
	LD	A, R	ED	5F		9	2		X	X	1	IFF*	0	A ← R transfere para o acumulador o conteúdo do registro de REFRESH
	LD	I, A	ED	47		9	2							I ← A transfere para o registro Vetor de Interrupção o conteúdo do acumulador
	LD	R, A	ED	4F		9	2							R ← A transfere para o registro de REFRESH o conteúdo do acumulador
	LDI		ED	A0		16	2				X	0	0	(DE) ← (HL), DE ← DE + 1, HL ← HL+1 e BC ← BC − 1 P/O = 1 se BC − 1 = 0, caso contrário: P/O = 0
	LDIR		ED	B0		21/16**	2				0	0	0	Repete até BC = 0: [(DE) ← (HL), DE ← DE + 1, HL ← HL + 1 e BC ← BC − 1]
	LDD		ED	A8		16	2				X	0	0	(DE) ← (HL), DE ← DE − 1, HL ← HL − 1 e BC − 1
	LDDR		ED	B8		21/16**	2				0	0	0	Repete até BC = 0: [(DE) ← (HL), DE ← DE − 1, HL ← HL − 1 e BC ← BC − 1]
	EX	DE,HL	EB			4	1							DE ←→ HL
	EX	AF,AF'	08			4	1							AF ←→ AF'
	EXX		D9			4	1							(BC, DE, HL) ←→ (BC', DE', HL')
	EX	(SP),HL	E3			19	1							H ←→ (SP + 1) e L ←→ (SP)
	EX	(SP),IX	DD	E3		23	2							IX_A ←→ (SP+1) e IX_B ←→ (SP)
	EX	(SP),IY	FD	E3		23	2							IY_A ←→ (SP+1) e IY_B ←→ (SP)
ARITMÉTICAS	ADD	A,r	10000 r			4	1	X	X	X	X	X	0	A ← A + r
	ADD	A,e	C6	e		7	2	X	X	X	X	X	0	A ← A + e
	ADD	A,(HL)	86			7	1	X	X	X	X	X	0	A ← A + (HL)
	ADD	A,(IX+e)	DD	86	e	19	3	X	X	X	X	X	0	A ← A + (IX+e)
	ADD	A,(IY+e)	FD	86	e	19	3	X	X	X	X	X	0	A ← A + (IY+e)
	ADC	A,r	10001 r			4	1	X	X	X	X	X	0	A ← A + r + CY
	ADC	A,e	CE	e		7	2	X	X	X	X	X	0	A ← A + e + CY

* O "flip-flop" que habilita a interrupção da CPU (IFF) é copiado do flag P/O.

**Apenas uma iteração.

TIPO	MNEMÔNICO	OPERANDO	CÓDIGO-OBJETIVO			PERÍODOS	TAMANHO	INDICADORES – FLAGS						DESCRIÇÃO
								CY	Z	S	P/O	Ac	N	
ARITMÉTICAS	ADC	A,(HL)	8E			7	1	X	X	X	X	X	0	A ← A + (HL) + CY
	ADC	A,(IX+e)	DD	8E	e	19	3	X	X	X	X	X	0	A ← A + (IX+e) + CY
	ADC	A,(IY+e)	FD	8E	e	19	3	X	X	X	X	X	0	A ← A + (IY+e) + CY
	SUB	r	10010 r			4	1	X	X	X	X	X	1	A ← A − r
	SUB	e	D6	e		7	2	X	X	X	X	X	1	A ← A − e
	SUB	(HL)	96			7	1	X	X	X	X	X	1	A ← A − (HL)
	SUB	(IX+e)	DD	96	e	19	3	X	X	X	X	X	1	A ← A − (IX+e)
	SUB	(IY+e)	FD	96	e	19	3	X	X	X	X	X	1	A ← A − (IY+e)
	SBC	A,r	10011 r			4	1	X	X	X	X	X	1	A ← A − r − CY
	SBC	A,e	DE	e		7	2	X	X	X	X	X	1	A ← A − e − CY
	SBC	A,(HL)	9E			7	1	X	X	X	X	X	1	A ← A − (HL) − CY
	SBC	A,(IX+e)	DD	9E	e	19	3	X	X	X	X	X	1	A ← A − (IX+e) − CY
	SBC	A,(IY+e)	FD	9E	e	19	3	X	X	X	X	X	1	A ← A − (IY+e) − CY
	INC	r	00r100			4	1	X	X	X	X	X	0	r ← r+1
	INC	(HL)	34			4	1	X	X	X	X	X	0	(HL) ← (HL) + 1
	INC	(IX+e)	DD	34	e	23	3	X	X	X	X	X	0	(IX+e) ← (IX+e) + 1
	INC	(IY+e)	FD	34	e	23	3	X	X	X	X	X	0	(IY+e) ← (IY+e) + 1
	DEC	r	00r101			4	1	X	X	X	X	X	1	r ← r−1
	DEC	(HL)	35			11	1	X	X	X	X	X	1	(HL) ← (HL) − 1
	DEC	(IX+e)	DD	35	e	23	3	X	X	X	X	X	1	(IX+e) ← (IX+e) − 1
	DEC	(IY+e)	FD	35	e	23	3	X	X	X	X	X	1	(IY+e) ← (IY+e) − 1
	ADD	HL,BC	09			11	1	X				?	0	HL ← HL + BC
	ADD	HL,DE	19			11	1	X				?	0	HL ← HL + DE
	ADD	HL,HL	29			11	1	X				?	0	HL ← HL + HL
	ADD	HL,SP	39			11	1	X				?	0	HL ← HL + SP
	ADC	HL,BC	ED	4A		15	2	X	X	X	X	?	0	HL ← HL + BC + CY
	ADC	HL,DE	ED	5A		15	2	X	X	X	X	?	0	HL ← HL + DE + CY
	ADC	HL,HL	ED	6A		15	2	X	X	X	X	?	0	HL ← HL + HL + CY
	ADC	HL,SP	ED	7A		15	2	X	X	X	X	?	0	HL ← HL + SP + CY
	SBC	HL,BC	ED	42		15	2	X	X	X	X	?	1	HL ← HL − BC − CY
	SBC	HL,DE	ED	52		15	2	X	X	X	X	?	1	HL ← HL − DE − CY
	SBC	HL,HL	ED	62		15	2	X	X	X	X	?	1	HL ← HL − HL − CY
	SBC	HL,SP	ED	72		15	2	X	X	X	X	?	1	HL ← HL − SP − CY
	ADD	IX,BC	DD	09		15	2	X				?	0	IX ← IX + BC
	ADD	IX,DE	DD	19		15	2	X				?	0	IX ← IX + DE
	ADD	IX,IX	DD	29		15	2	X				?	0	IX ← IX + IX
	ADD	IX,SP	DD	39		15	2	X				?	0	IX ← IX + SP
	ADD	IY,BC	FD	09		15	2	X				?	0	IY ← IY + BC
	ADD	IY,DE	FD	19		15	2	X				?	0	IY ← IY + DE
	ADD	IY,IY	FD	29		15	2	X				?	0	IY ← IY + IY

TIPO	MNEMÔNICO	OPERANDO	CÓDIGO-OBJETIVO	PERÍODOS	TAMANHO	INDICADORES — FLAGS						DESCRIÇÃO
						CY	Z	S	P/O	Ac	N	
ARITMÉTICAS	ADD	IY,SP	FD 39	15	2	X				?	0	IY ← IY + SP
	INC	BC	03	6	1							BC ← BC + 1
	INC	DE	13	6	1							DE ← DE + 1
	INC	HL	23	6	1							HL ← HL + 1
	INC	SP	33	6	1							SP ← SP + 1
	INC	IX	DD 23	10	2							IX ← IX + 1
	INC	IY	FD 23	10	2							IY ← IY + 1
	DEC	BC	0B	6	1							BC ← BC − 1
	DEC	DE	1B	6	1							DE ← DE − 1
	DEC	HL	2B	6	1							HL ← HL − 1
	DEC	SP	3B	6	1							SP ← SP − 1
	DEC	IX	DD 2B	10	2							IX ← IX − 1
	DEC	IY	FD 2B	10	2							IY ← IY − 1
	DAA		27	4	1	X	X	X	X	X		Ajuste decimal, assume que o acumulador, contém a soma ou diferença de dois dígitos BCD
DESVIO	JP	end	C3 ← end →	10	3							PC ← end
	JP	NZ,end	C2 ← end →	10	3							Se Z = 0,
	JP	Z,end	CA ← end →	10	3							Se Z = 1,
	JP	NC,end	D2 ← end →	10	3							Se CY = 0,
	JP	C,end	DA ← end →	10	3							Se CY = 1, } [PC ← end]
	JP	PO,end	E2 ← end →	10	3							Se P/O = 0,
	JP	PE,end	EA ← end →	10	3							Se P/O = 1,
	JP	P,end	F2 ← end →	10	3							Se S = 0,
	JP	M,end	FA ← end →	10	3							Se S = 1,
	JR	e	18 e-2	12	2							PC ← PC + 2 + e-2
	JR	C,e	38 e-2	7/12	2							Se CY = 1,
	JR	NC,e	30 3-2	7/12	2							Se CY = 0, } [PC ← PC + e]
	JR	Z,e	28 e-2	7/12	2							Se Z = 1,
	JR	NZ,e	20 e-2	7/12	2							Se Z = 0,
	JP	(HL)	E9	4	1							PC ← HL
	JP	(IX)	DD E9	8	2							PC ← IX
	JP	(IY)	FD E9	8	2							PC ← IY
	DJNZ	e	10 e-2	8/13	2							B ← B-1; Se B≠0: [PC ← PC + e]
	CALL	end	CD ← end →	17	3							(SP-1) ← PC_A; (SP) ← PC_B; SP-2; PC ← end
	CALL	NZ,end	C4 ← end →	10/17	3							Se Z = 0,
												} [(SP-1) ← PC_A; (SP) ← PC_B; SP ← SP-2; PC ← end]
	CALL	Z,end	CC ← end →	10/17	3							Se Z = 1,

TIPO	MNEMÔNICO	OPERANDO	CÓDIGO-OBJETIVO	PERÍODOS	TAMANHO	INDICADORES – FLAGS: CY	Z	S	P/O	Ac	N	DESCRIÇÃO
DESVIO	CALL	NC,end	D4 ← end →	10/17	3							Se CY = 0,
	CALL	C,end	DC ← end →	10/17	3							Se CY = 1,
	CALL	PO,end	E4 ← end →	10/17	3							Se P/O = 0, [(SP-1) ← PC_A; (SP) ← PC_B; SP ← SP-2; PC ← end]
	CALL	PE,end	EC ← end →	10/17	3							Se P/O ← 1,
	CALL	P,end	F4 ← end →	10/17	3							Se S = 0,
	CALL	M,end	FC ← end →	10/17	3							Se S = 1,
	RET		C9	10	1							PC_B ← (SP); PC_A ← (SP+1); SP ← SP+2
	RET	NZ	C0	5/11	1							Se Z = 0,
	RET	Z	C8	5/11	1							Se Z = 1,
	RET	NC	D0	5/11	1							Se CY = 0,
	RET	C	D8	5/11	1							Se CY = 1, [PC_B ← (SP); PC_A ← (SP+1);
	RET	PO	E0	5/11	1							Se P/O = 0, SP ← SP+2]
	RET	PE	E8	5/11	1							Se P/O = 1,
	RET	P	F0	5/11	1							Se S= 0,
	RET	M	F8	5/11	1							Se S = 1,
	RETI		ED 4D	14	2							Retorno de interrupção
	RETN		ED 45	14	2							Retorno de interrupção não mascarada
	RST	b	11b111	11	1							(SP-1) ← PC_A; (SP-2) ← PC_B; SP ← SP-2; PC ← 8 × b
LÓGICAS	AND	r	10000 r	4	1	0	X	X	X	1	0	A ← A ∧ r
	AND	e	E6 e	7	2	0	X	X	X	1	0	A ← A ∧ e
	AND	(HL)	A6	7	1	0	X	X	X	1	0	A ← A ∧ (HL)
	AND	(IX+e)	DD A6 e	19	3	0	X	X	X	1	0	A ← A ∧ (IX+e)
	AND	(IY+e)	FD A6 e	19	3	0	X	X	X	1	0	A ← A ∧ (IY+e)
	OR	r	10110 r	4	1	0	X	X	X	1	0	A ← A ∨ r
	OR	e	F6 e	7	2	0	X	X	X	1	0	A ← A ∨ e
	OR	(HL)	B6	7	1	0	X	X	X	1	0	A ← A ∨ (HL)
	OR	(IX+e)	DD B6 e	19	3	0	X	X	X	1	0	A ← A ∨ (IX+e)
	OR	(IY+e)	FD B6 e	19	3	0	X	X	X	1	0	A ← A ∨ (IY+e)
	XOR	r	10101 r	4	1	0	X	X	X	1	0	A ← A ∀ r
	XOR	e	EE e	7	2	0	X	X	X	1	0	A ← A ∀ e
	XOR	(HL)	AE	7	1	0	X	X	X	1	0	A ← A ∀ (HL)
	XOR	(IX+e)	DD AE e	19	3	0	X	X	X	1	0	A ← A ∀ (IX+e)
	XOR	(IY+e)	FD AE e	19	3	0	X	X	X	1	0	A ← A ∀ (IY+e)
	CP	r	10111 r	4	1	X	X	X	X	X	1	A − r

TIPO	MNEMÔNICO	OPERANDO	CÓDIGO-OBJETIVO	PERÍODOS	TAMANHO	INDICADORES — FLAGS						DESCRIÇÃO
						CY	Z	S	P/O	Ac	N	
LÓGICAS	CP	e	FE e	7	2	X	X	X	X	X	1	A – e
	CP	(HL)	BE	7	1	X	X	X	X	X	1	A – (HL)
	CP	(IX+e)	DD BE e	19	3	X	X	X	X	X	1	A – (IX+e)
	CP	(IY+e)	FD BE e	19	3	X	X	X	X	X	1	A (IY+e)
	RLCA		07	4	1	X				0	0	CY ← 7 … 0 (rotação para a esquerda)
	RR	r	CB 0001 r	8	2	X	X	X	X	0	0	7 … 0 → CY — r, (HL), (IX+e), (IY+e) — rotação para a direita, através do VAI UM
	RR	(HL)	EC 1E	15	2	X	X	X	X	0	0	
	RR	(IX+e)	DD CB e 1E	23	4	X	X	X	X	0	0	
	RR	(IY+e)	FD CB e 1E	23	4	X	X	X	X	0	0	
	SLA	r	CB 00100 r	8	2	X	X	X	X	0	0	CY ← 7 … 0 ← 0 — r, (HL), (IX+e), (IY+e) — deslocamento para a esquerda
	SLA	(HL)	CB 26	15	X	X	X	X	X	0	0	
	SLA	(IX+e)	DD CB e 26	23	4	X	X	X	X	0	0	
	SLA	(IY+e)	FD CB e 26	23	4	X	X	X	X	0	0	
	SRA	r	CB 00101 r	8	2	X	X	X	X	0	0	7 … 0 → CY — r, (HL), (IX+e), (IY+e) — deslocamento para a direita preservando o bit 7
	SRA	(HL)	CB 2E	15	2	X	X	X	X	0	0	
	SRA	(IX+e)	DD CB e 2E	23	4	X	X	X	X	0	0	
	SRA	(IY+e)	FD CB e 2E	23	4	X	X	X	X	0	0	

TIPO	MNEMÔNICO	OPERANDO	CÓDIGO-OBJETIVO	PERÍODOS	TAMANHO	INDICADORES — FLAGS						DESCRIÇÃO
						CY	Z	S	P/O	Ac	N	
LÓGICAS	SRL	r	CB 00111 r	8	2	X	X	X	X	0	0	0 → [7 … 0] → CY — r, (HL), (IX+e), (IY+e) — deslocamento para a direita
	SRL	(HL)	CB 3E	15	2	X	X	X	X	0	0	
	SRL	(IX+e)	DD CB e 3E	23	4	X	X	X	X	0	0	
	SRL	(IY+e)	FD CB e 3E	23	4	X	X	X	X	0	0	
	RLD		EB 6F	18	2		X	X	X	0	0	A [7 4 3 0] (HL) [7 4 3 0] — deslocamento para a esquerda
	RRD		ED 67	18	2		X	X	X	0	0	A [7 4 3 0] (HL) [7 4 3 0] — deslocamento para a direita
	CPL		2F	4	1					1	1	A ← Ā complemento a um
	NEG		ED 44	8	2	X	X	X	X	X	1	A ← Ā+1 complemento a dois
	CCF		3F	4	1	X				?	0	CY ← $\overline{CY}$
	SCF		37	4	1	1				0	0	CY ← 1
	CPI		ED A1	16	2		X	X	X	X	1	A − (HL) somente os flags são alterados; HL ← HL+1; BC ← BC-1
	CPIR		ED B1	21/16*	2		X	X	X	X	1	Repete até A = (HL) ou BC = 0: [A (HL): somente os flags são alterados; HL ← HL+1; BC ← BC-1]
	CPD		ED A9	16	2		X	X	X	X	1	A − (HL), somente os flags são alterados; HL ← HL−1; BC ← BC−1
	CPDR		ED B9	21/16*	2		X	X	X	X	1	Repete até A = (HL) ou BC = 0, [A−(HL); somente os flags são alterados; HL ← HL−1; BC ← BC−1]
	BIT	b,r	CB 01 b r	8	2		X	?	?	1	0	Z ← $\overline{r}_b$
	BIT	b,(HL)	CB 01 b 110	12	2		X	?	?	1	0	Z ← $\overline{(HL)}_b$
	BIT	b,(IX+e)	DD CB e 01b110	20	4		X	?	?	1	0	Z ← $\overline{(IX+e)}_b$
	BIT	b,(IY+e)	FD CB e 01b110	20	4		X	?	?	1	0	Z ← $\overline{(IY+e)}_b$

* Apenas uma iteração.

TIPO	MNEMÔNICO	OPERANDO	CÓDIGO-OBJETIVO	PERÍODOS	TAMANHO	CY	Z	S	P/O	Ac	N	DESCRIÇÃO
LÓGICAS	RLA		17	4	1	X				0	0	CY 7 0 — rotação para a esquerda, através do VAI UM
	RRCA		0F	4	1	X				0	0	7 0 CY — rotação para a direita
	RRA		1F	4	1	X				0	0	7 0 CY — rotação para a direita, através do VAI UM
	RLC	r	CB 00000 r	8	2	X	X	X	X	0	0	CY 7 0 — r, (HL), (IX+e), (IY+e) — rotação para a esquerda
	RLC	(HL)	CB 06	15	2	X	X	X	X	0	0	
	RLC	(IX+e)	DD CB e 06	23	4	X	X	X	X	0	0	
	RLC	(IY+e)	FD CB e 06	23	4	X	X	X	X	0	0	
	RL	r	CB 00010 r	8	2	X	X	X	X	0	0	CY 7 0 — r, (HL), (IX+e), (IY+e) — rotação a esquerda através do VAI UM
	RL	(HL)	CB 16	15	2	X	X	X	X	0	0	
	RL	(IX+e)	DD CB e 16	23	4		X	X	X	0	0	
	RL	(IY+e)	FD CB e 16	23	4	X	X	X	X	0	0	

TIPO	MNEMÔNICO	OPERANDO	CÓDIGO-OBJETIVO				PERÍODOS	TAMANHO	INDICADORES – FLAGS						DESCRIÇÃO
									CY	Z	S	P/O	Ac	N	
LÓGICAS	RRC	r	CB	00001 r			8	2	X	X	X	X	0	0	r — 7 … 0 → CY, rotação para a direita
	RRC	(HL)	CB	0E			15	2	X	X	X	X	0	0	(HL)
	RRC	(IX+e)	DD	CB	e	0E	23	4	X	X	X	X	0	0	(IX+e)
	RRC	(IY+e)	FD	CB	e	0E	23	4	X	X	X	X	0	0	(IY+e)
	SET	b,r	CB	11 b r			8	2							$r_b \leftarrow 1$
	SET	b,(HL)	CB	11 b 110			15	2							$(HL)_b \leftarrow 1$
	SET	b,(IX+e)	DD	CB	e	11b110	23	4							$(IX+e)_b \leftarrow 1$
	SET	b,(IY+e)	FD	CB	e	11b110	23	4							$(IY+e)_b \leftarrow 1$
	RES	b,r	CB	10 b r			8	2							$r_b \leftarrow 0$
	RES	b,(HL)	CB	10 b 110			15	2							$(HL)_b \leftarrow 0$
	RES	b,(IX+e)	DD	CB	e	10b110	23	4							$(IX+e)_b \leftarrow 0$
	RES	b,(IY+e)	FD	CB	e	10b110	23	4							$(IY+e)_b \leftarrow 0$
PILHA, E/S e OUTRAS	PUSH	BC	C5				11	1							$(SP-2) \leftarrow C$; $(SP-1) \leftarrow B$; $SP \leftarrow SP-2$
	PUSH	DE	D5				11	1							$(SP-2) \leftarrow E$; $(SP-1) \leftarrow D$; $SP \leftarrow SP-2$
	PUSH	HL	E5				11	1							$(SP-2) \leftarrow L$; $(SP-1) \leftarrow H$; $SP \leftarrow SP-2$
	PUSH	AF	F5				11	1							$(SP-2) \leftarrow F$; $(SP-1) \leftarrow A$; $SP \leftarrow SP-2$
	PUSH	IX	DD	E5			15	2							$(SP-2) \leftarrow IX_B$; $(SP-1) \leftarrow IX_A$; $SP \leftarrow SP-2$
	PUSH	IY	FD	E5			15	2							$(SP-2) \leftarrow IY_B$; $(SP-1) \leftarrow IY_A$; $SP \leftarrow SP-2$
	POP	BC	C1				10	1							$C \leftarrow (SP)$; $B \leftarrow (SP+1)$; $SP \leftarrow SP+2$
	POP	DE	D1				10	1							$E \leftarrow (SP)$; $D \leftarrow (SP+1)$; $SP \leftarrow SP+2$
	POP	HL	E1				10	1							$L \leftarrow (SP)$; $H \leftarrow (SP+1)$; $SP \leftarrow SP+2$
	POP	AF	F1				10	1							$F \leftarrow (SP)$; $A \leftarrow (SP+1)$; $SP \leftarrow SP+2$
	POP	IX	DD	E1			14	2							$IX_B \leftarrow (SP)$; $IX_A \leftarrow (SP+1)$; $SP \leftarrow SP+2$
	POP	IY	FD	E1			14	2							$IY_B \leftarrow (SP)$; $IY_A \leftarrow (SP+1)$; $SP \leftarrow SP+2$
	DI		F3				4	1							Desabilita interrupções
	EI		FB				4	1							Habilita interrupção
	IM	0	ED	46			8	2							Coloca o UPC no modo 0 de interrupção
	IM	1	ED	56			8	2							Coloca o UCP no modo 1 de interrupção
	IM	2	ED	5E			8	2							Coloca o UPC no modo 2 de interrupção
	NOP		00				4	1							Nenhuma operação é executada
	HALT		76				4	1							Para a UCP

TIPO	MNEMÔNICO	OPERANDO	CÓDIGO-OBJETIVO		PERÍODOS	TAMANHO	INDICADORES – FLAGS						DESCRIÇÃO
							CY	Z	S	P/O	Ac	N	
PILHA, E/S e OUTRAS	IN	A,(e)	DB	e	10	2							A ← (porta E/S e)
	IN	r,(C)	ED	01r000	11	2		X	X	X	X	0	r ← (porta E/S indicada pelo registro C)
	INI		ED	A2	15	2		X	?	?	?	1	(HL) ← (porta de E/S indicada pelo registro C); B ← B−1, HL ← HL+1
	INIR		ED	B2	20/15*	2		1	?	?	?	1	Repete até B=0: [(HL) ← (porta de E/S indicada pelo registro C); B ← B − 1; HL ← HL+1]
	IND		ED	AA	15	2		X	?	?	?	1	(HL) ← (porta de E/S indicada pelo registro C); B ← B−1; HL ← HL−1
	INDR		ED	BA	20/15*	2		1	?	?	?	1	Repete até B=0: [(HL) ← (porta de E/S indicada pelo registro C); B ← B−1; HL ← HL−1]
	OUT	(e),A	D3	e	11	2							(porta de E/S e) ← A
	OUT	(C),r	ED	01r001	12	2							(porta de E/S indicada pelo registro C) ← r
	OUTI		ED	A3	15	2		X	?	?	?	1	(porta de E/S indicada pelo registro C) ← (HL); B ← B−1; HL ← HL+1
	OTIR		ED	B3	20/15*	2		1	?	?	?	1	Repete até B=0: [(porta de E/S indicada pelo registro C) ← (HL); B ← B−1; HL ← HL+1]
	OUTD		ED	AB	15	2		X	?	?	?	1	(Porta E/S indicada pelo registro C) ← (HL); B ← B−1; HL ← HL−1
	OTDR		ED	BB	20/15*	2		1	?	?	?	1	Repete até B=0: [(porta de ES indicada pelo registro C) ← (HL); B ← B−1; HL ← HL−1]

* Apenas uma iteração.

Microcomputadores

Arquitetura • Projeto • Programação

PAULO BIANCHI
MILTON BEZERRA

O texto se destina aos leitores que já sabem o que é um computador e o que é um programa. Não é necessário nenhum conhecimento de eletrônica. Visa ao aprofundamento dos conhecimentos relativos à construção de computadores e programação a nível de linguagem *assembler* e de máquina.

Os profissionais e estudantes de computação, bem como os entusiastas do computador pessoal encontrarão aqui um meio de satisfazer a sua ânsia de aprofundamento.

MAIS UM LANÇAMENTO
DE
LIVROS TÉCNICOS E CIENTÍFICOS EDITORA S.A.

ISBN: 85-216-0321-5